



NAVAL
POSTGRADUATE
SCHOOL

MONTEREY, CALIFORNIA

THESIS

AUTO-CONFIGURATION OF CISCO ROUTERS WITH
APPLICATION SOFTWARE

by

Alexandre Barcellos Prado

September 2003

Thesis Advisor:	Geoffrey Xie
Second Reader:	John Gibson

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE September 2003	3. REPORT TYPE AND DATES COVERED Master's Thesis	
4. TITLE AND SUBTITLE: Auto-Configuration of Cisco Routers with Application Software			5. FUNDING NUMBERS	
6. AUTHOR(S) Prado, Alexandre				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			10. SPONSORING/MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			12b. DISTRIBUTION CODE Statement A	
13. ABSTRACT The context of this research is to facilitate the control of routers with the Server and Agent based Active Network Management (SAAM), to optimize allocation of network resources, and to satisfy user Quality of Service (QoS) requirements. The SAAM network determines the Quality of Service parameters based on current network conditions and user requirements. These parameters are dynamic, so they must be uploaded into the Cisco routers to take effect. The focus is on determining the most efficient way of communicating between the Cisco routers and the SAAM system. This is accomplished by several key components of the proxy-based solution as the first step for integrating with a legacy network. This thesis develops methods and application software to automatically update the configurations of Cisco routers based on the current network condition. These methods are required by any solution that resolves to upgrade the existing legacy network to provide QoS without expensive equipment replacement. As a result users are provided with a network they can expect to function properly.				
14. SUBJECT TERMS Cisco, IOS, Console, Ethernet, Telnet, SSH, QoS, Perl, socket, HyperACCESS, API, HAPI, Visual Basic, C++.			15. NUMBER OF PAGES 93	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL	

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

**AUTO-CONFIGURATION OF CISCO ROUTERS WITH APPLICATION
SOFTWARE**

Alexandre B. Prado
Major, Brazilian Air Force
B.S., Brazil's Air Force Academy, 1985

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

**NAVAL POSTGRADUATE SCHOOL
September 2003**

Author: Alexandre Barcellos Prado

Approved by: Geoffrey Xie
Thesis Advisor

John H. Gibson
Second Reader

Peter J. Denning
Chairman, Department of Computer
Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The context of this research is to facilitate the control of routers with the Server and Agent based Active Network Management (SAAM), to optimize allocation of network resources, and to satisfy user Quality of Service (QoS) requirements. The SAAM network determines the Quality of Service parameters based on current network conditions and user requirements. These parameters are dynamic, so they must be uploaded into the Cisco routers to take effect. The focus is on determining the most efficient way of communicating between the Cisco routers and the SAAM system. This is accomplished by several key components of the proxy-based solution as the first step for integrating with a legacy network.

This thesis develops methods and application software to automatically update the configurations of Cisco routers based on the current network condition. These methods are required by any solution that resolves to upgrade the existing legacy network to provide QoS without expensive equipment replacement. As a result users are provided with a network they can expect to function properly.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I.	INTRODUCTION	1
A.	CONCEPTION	1
B.	THE SCOPE OF THE THESIS	3
II.	BACKGROUND	5
A.	ELEMENTS OF DYNAMIC MANAGEMENT IN NETWORK QUALITY OF SERVICE	5
1.	Automatic and Timely Update of a Router Configuration	6
2.	Automatic and Timely Collecting of Router State Information	8
3.	Automatic and Timely Communication Between Different Managing Agents	9
B.	SERVER AND AGENT-BASED ACTIVE NETWORK MANAGEMENT ..	11
1.	Architecture	12
a.	SAAM Server	13
b.	SAAM Proxy	13
C.	CISCO ROUTER CONFIGURATION	16
1.	IOS and IOS Command Line Interface	16
D.	THE CISCO ROUTER	20
1.	Cisco Console Port	22
a.	HyperTerminal	23
b.	HyperACCESS	24
2.	Remote Cisco Configuration	25
a.	Telnet	26
b.	Perl	28
III.	COMMUNICATION USING PERL SCRIPT	31
A.	SIMPLE NETWORK MANAGEMENT PROTOCOL	31
1.	Configuration Experiment	33
a.	Uploading a Router Configuration Using Perl Script	36
b.	Collection the Router Configuration Using Perl Script	38
c.	Automatic and Timely Communication Between Different Managing Agents	40
d.	Code and Answers	42
B.	CONCLUSION	46
IV.	ROUTER CONFIGURATION USING CONSOLE PORT	49
A.	HYPERACCESS	49
B.	SCRIPT LANGUAGES	50
1.	Configuration Experiments	50
a.	Upload of a Router Configuration Using HyperACCESS with VBScript	51

b.	<i>Collection of Router Configuration Information Using HyperACCESS with JavaScript</i>	<i>57</i>
C.	CUSTOM-WRITTEN PROGRAMS	61
1.	Microsoft Visual Basic	62
a.	<i>The Code in Visual Basic</i>	<i>63</i>
2.	Microsoft Visual C++	67
V.	CONCLUSION	69
A.	THE PROBLEM	69
B.	SOLVING	70
C.	FUTURE WORK	72
	LIST OF REFERENCES	75
	INITIAL DISTRIBUTION LIST	77

LIST OF FIGURES

Figure 1. Example of a Network.	16
Figure 2. Lay-out of the Demonstration Network.	34
Figure 3. Configuration of Router Using Telnet.	39
Figure 4. Upload of the Configuration Data with Open Perl IDE.	40
Figure 5. Collection of the data with Open Perl IDE.	41
Figure 6. Receipt of a Message from Another Socket.	43
Figure 7. Transport of a String to Another Socket.	44
Figure 8. HyperAccess Linked with the Router Showing the Result of the Elapsed Time for the Configuration Update Interaction Using a VBScript Application. ..	55
Figure 9. HyperACCESS linked with the Router Displaying the Results of the Queue Status Query.	59
Figure 10 Visual Basic Accessing the HyperACCESS Library. ...	62

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

Table 1.	Table of Metrics in a Network.	10
----------	-------------------------------------	----

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENT

The human being is an animal that can easily adapt to many environments. I believe this because after spending only two years three months and twenty days of my life in another culture, I quickly adapted too many new experiences of living.

Everything began with the expectation of receiving a master's degree and in living in a country that has the most powerful economy in the world. The assumption that my time is finishing comes with the completion of the thesis.

I feel like a winner, but I admit that many people participated in my victory. First, I would like to thank God for blessing me everyday. Next I would like to extend my gratitude to Carla Ribeiro, Marlene Queiroz, Vo Evanyr and Andrea Silva for their support during some difficult moments, to my father for being proud of his son, to my child Eduardo for giving me a new sense of my life. Additionally, I want to thank my English instructor, Beth Summe for guiding me through the first step of communication in the United States, to the faculty and staff of the Department of Computer Sciences, especially my adviser Professor Geoffrey Xie and my second reader Professor John Gibson for their high level of professionalism. Finally I want to say thanks for the friendships, good times and the fantastic American people for their generosity and kindness.

This master will be eternalized in my mind and in my heart as a frontier that I overcame and as a challenge to know life and myself.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. CONCEPTION

Government agencies, companies, and individuals increasingly rely on their networks to run business-critical applications, such as telephony and video communications. Therefore, ensuring the provision of established and accurate network quality of service (QoS) levels is necessary.

Although characteristics of QoS vary by industry and government requirements, the term QoS generally describes a network's ability to function with a high level of reliability, to operate with minimal downtime, and to ensure that the bandwidth-intensive applications, such as voice and video communications, perform effectively.

Traditionally, a number of different devices are used at the LAN-WAN interface. These devices include data service units (DSU) for basic connectivity, WAN probes and routers to direct data traffic, and other solutions for monitoring functions (particularly for Service Level Agreements).

Of these devices, the routers are the most popular way for interconnecting the network to provide a path of communication with external hosts.

Quality of service is a critical starting point for developing methods and application software to update the configurations of Cisco routers autonomously, based on the current network condition. The methods are required for any

solution that attempts to upgrade the existing legacy network. This, therefore, establishes a QoS without expensive equipment replacement while providing users with a network that function properly.

To satisfy the QoS needs many products are being developed to provide routers without modifying the IOS. If the IOS is proprietary software, as is the case with Cisco products, obtaining the IOS source code or arranging for a special IOS update is time-consuming and potentially costly.

Based on this information, the Server and Agent based Active Network Management system (SAAM) was created. The goal of the SAAM is to optimize allocation of the network resources to support the user Quality of Service (QoS) requirements.

The SAAM network determines the Quality of Service parameters based on user requirements and allocates network resources accordingly. These resources allocations are dynamic and must be uploaded into the Cisco routers for them to take effect.

Therefore, understanding the parameters of the Cisco IOS is essential when the Server is configuring or externally controlling the router. For this approach, an autonomous method must be created to communicate with the routers, the core devices of a real commercial and pre-installed network, and the QoS parameters such as flow definitions and the routing table entries.

This communication must be in two directions. The first direction makes it possible for the SAAM server to

extract information from the Cisco router or to provide configuration settings to the router. The other simultaneously connects the SAAM server to another server to upload pertinent information to the entire network.

B. THE SCOPE OF THE THESIS

The goal of this research is to develop a method of remotely and autonomously configuring a router's interface parameters to establish and maintain required service levels. This thesis will consider the software SAAM architecture, with its relevant applications, standards and protocols. The thesis will further identify the actions that must be taken, either by the server or the router, and the information that must be exchanged between the server and the router to dynamically and adaptively configure the router so that the desired QoS level is achieved and maintained.

In addition, the thesis will demonstrate the benefits of using software such as HyperACCESS to autonomously communicate with the router. HyperACCESS, a product of Hilgraeve, provides an Application Programming Interface (API) that can be activated from application programs, developed in JavaScript or VBScript. The API enables the server to open a session with a router and to configure its interface parameters.

Data from tests performed in support of this thesis will be analyzed to verify the effectiveness of the present methodology. A successful adaptive and autonomous configuration of a router test-bed will validate this research's contribution of creating an efficient mode of

dynamic communication between a router and the Server and Agent based Active Network Management system.

II. BACKGROUND

A. ELEMENTS OF DYNAMIC MANAGEMENT IN NETWORK QUALITY OF SERVICE

The communication capability of a corporate network forms the information backbone of any successful organization. Such a network transports a multitude of applications and data, including high-quality video and delay-sensitive data such as real-time voice. Bandwidth-intensive applications not only press the limits of network capabilities and resources but they also complement, add value, and enhance every business process.

Networks must provide secure and dependable services, often with a guarantee of quality. Achieving the required quality of service (QoS), a set of measurable network characteristics that must be managed across network resources, forms the foundation for a successful end-to-end business solution. QoS is the set of measurable characteristics of a network, which must be managed across network resources.

The first step in deploying a consistent, supportable QoS is to identify and categorize the network traffic generated by each application. Access control lists (ACLs) are the most commonly used tools for identifying and controlling network traffic. The ACLs use information from Layer 3 (IP address) and Layer 4 (TCP/UDP port numbers) to categorize traffic. However, using ACLs alone to manage QoS rapidly increases the size and the number of ACLs required for a network. Furthermore, they cannot easily identify all applications because some port numbers are not standardized, and therefore cannot be associated with a

particular application. In addition, custom applications use non-standardized ports and therefore cannot be generally categorized.

To achieve the desired level of application performance, establishing the initial QoS policy is required. Cisco's QoS constructs enable the network manager to balance the QoS policy variables (bandwidth, delay, and jitter and packet loss), and to determine the policies for delivering the desired application performance.

1. Automatic and Timely Update of a Router Configuration

Network devices need to be programmed with the correct set of features and parameters to implement the established policy. While QoS functionality is rich in features, the process of an effective implementation requires a thorough study of the network for which the QoS policy is to be employed.

Ideally not requiring any modification to the Internetworking Operating System (IOS), the software of the Cisco router, is preferred for such uploads. This is because the IOS is proprietary software. This software is potentially costly to obtain the IOS source code from Cisco or arrange for a special IOS update by Cisco. Therefore, without automation, the QoS configuration challenge can be very complex.

To manage the packet delay along each link from source to destination, the data capacity of each link and the packet loss parameters of a network need to be timely. This is principally because the traffic and the congestion in a network can change very quickly. Therefore, a router

must maintain its router table with efficiency, following the set of rules used by the router to reach the desired QoS.

For a better understanding of this approach it is necessary to imagine a network configured to satisfy QoS constraints in its data transmission. The network needs to establish a reserved-bandwidth path between end systems or hosts on either side to ensure the availability of adequate resources.

The need for network resource reservations differs for data traffic versus real-time traffic, which is live voice or video information. Suppose this need is restricted only to real-time traffic. Since such traffic is an almost constant flow of information, the network must insure in its movement of the information. Some guarantee must be provided that service between the hosts will not vary, because such variance can cause a loss of synchronization or insufficient bandwidth, which then may cause delay variations or information loss due to congestion.

The Resource Reservation Protocol (RSPV), an IP service available in the Cisco router, provides some mechanisms that enable real-time traffic to reserve resources necessary for consistent latency. For example, to enable RSPV on an interface, the command `ip rspv bandwidth [interface-kbps] [single-flow-kbps]` must be entered. This command starts RSPV and sets the bandwidth and single-flow limits.

This thesis demonstrates the ability of a software application to cause the execution of this reservation command without modifying the proprietary IOS or manual

entry on an interfaced keyboard. This application may help to reduce the time and cost of manually managing network service quality.

2. Automatic and Timely Collecting of Router State Information

Collecting data on bandwidth, reliability, packet counts by interface, and other essential information about the health of a network is crucial for maintaining the status the network. Automating this collection of information from the network's routers is very important for maintaining the QoS levels of the network. An approach similar to that given above may be used to automate this task.

The routers have available information generated internally, the traffic load offered determines great deals of which. This load must share the available network resources and must place constraints on the minimum services required, which is determined by the traffic type, such as real-time traffic. This traffic type requires constant bit-rates or an interactive data application whose traffic comes in sporadic bursts.

If a host is requesting a specific quality of service the network devices need to maintain the right mix of features and set of parameter values to implement the necessary policy. The configuration that the QoS determinates must not be time-consuming. The information acquired from the routers needs to be available all the time compare with the desired QoS thresholds.

Thus, the collection of router information must be automated to support responsive QoS management.

For example, after the RSVP protocol parameters are configured to implement the desired network resource policy, verifying the resulting RSVP settings is possible by using commands such as **show ip rsvp interface** [type number] and **show ip rsvp installed** [type number]. These commands display RSVP-related interface, filters, and bandwidth information. These results can be checked in a timely manner if the collection of the values is automated to verify any change that has occurred in the established parameter values. This automated process makes it possible to check the router without a change in the proprietary of operating system or manual entry by the network administrator, saving both time and money.

3. Automatic and Timely Communication Between Different Managing Agents

Routers are capable of supporting multiple independent routing protocols and maintaining routing tables for several routed protocols. When a router algorithm updates a routing table, its primary objective is to determine the best information to include in the table, based on one of more of the criteria shown in Table 1 (below). When the desire is to support a particular Quality of Service, many parameters should be considered. Therefore, many kinds of networks each with its own characteristics must be managed.

The metrics most commonly used by routers are as follows:

- **Bandwidth** Data capacity of a link
- **Delay** Length of time required to move a packet along each link from source to destination
- **Load** The amount of activity on a network resource such as a router or a link
- **Reliability** Usually refers to the error rate or packet loss of each network link
- **Hop count-destination** The number of routers a packet travels through before reaching its destination
- **Ticks** The delay on a data link using IBM PC clocks ticks (approximately 55 milliseconds)
- **Cost** An arbitrary value, usually based on bandwidth, monetary expenses, or other measurements, assigned by network administrator

Table 1. Table of Metrics in a Network

In this project, software of the agents is associated with the routers to help manage the router configurations by proving dynamic control over resources. In this way, the agents support internetwork connectivity and help maintain reliable performance and resource flexibility. A proxy agent must communicate with its associated router and

other proxies in a timely, reliable, and secure manner. Based on this approach, finding an effective way to manage all the agents included in the network is essential to communicating effectively among the agents. The figure below shows one such example for using agents.

The router agents automatically collect all the pertinent configuration and performance information from the router, and then send it to the server proxy. After analyzing the traffic situation and received data, the server proxy sends any necessary configuration change directives of proxies to the router in order to provide a continuous QoS.

If the **show ip rsvp** commands, described above, were used to collect information from a router to verify the current RSVP protocol settings, and if the information showed that the bandwidth of that router is below that required to sustain the desired QoS, the proxy agent will send this information to the server proxy. The server then performs a calculation of the required parameter values and generates a set of commands to be forwarded to the pertinent router agents to adjust the rsvp bandwidth appropriately.

B. SERVER AND AGENT-BASED ACTIVE NETWORK MANAGEMENT

The Server and Agent-Based Active Network Management (SAAM) system is a research project under the DARPA-funded Next Generation Internet (NGI) initiative that was initiated in the Naval Postgraduate School in 1998. It is a distributed control system, which has, as a main goal, the functionality to optimize allocation of the network

resources to support user Quality of Service (QoS) requirements. The system provides an efficient solution for managing the support of multimedia applications in the network.

1. Architecture

The SAAM architecture consists of the SAAM Server that controls the SAAM system, the SAAM-enabled routers, and the SAAM proxies. The Server makes decisions based on the information it collects from SAAM-enabled routers and SAAM proxies. This information gives the server a global picture of the specific region of the network it controls. The SAAM-enabled routers are lightweight routers, in that they only forward the packets to their destination address based on routing decisions made by the SAAM Server.

To make the SAAM scalable for large networks, its Servers are organized into a hierarchy. In the first level, the network is divided into autonomous regions, called the SAAM regions. One SAAM Server, for each region. A central management server controlling the entire network, which makes a single point of reference possible through the media applications that coordinate a required Quality of Service guarantee, supports the regional servers. Instead of negotiating with local routers or regional servers for end-to-end service that cannot be guaranteed, the service request is always sent to the appropriate SAAM server, which decides how to maintain the QoS level for the requested end-to-end service.

a. SAAM Server

The SAAM server periodically collects information from its supported SAAM routers and proxies while monitoring the actual QoS parameters of the controlled network. This information is aggregating into a read-to-use database of useful paths. The database is called the Path Information Base (PIB). By using this database, the SAAM server can implement network functions, such as routing and re-routing of real-time flows, which are required to provide Differentiated Service.

b. SAAM Proxy

A SAAM Proxy handles actual data traffic from the client applications, acting upon the information received from the routers. The requests for data transport support are forward to the SAAM Server for approval. The Server responds to each request by assigning a path over which to forward the data or by denying the request. When the path is first created, the Server sends route-update messages to all the routers in the path directs them to install appropriate entries to their routing tables, creating the packet-forwarding path.

Each proxy is responsible for periodically sending status to the Server. These status messages allow the Server to maintain a precise view of the entire network. The Server then has the capacity to make an accurate routing decision. The information follows the hierarchical structure of the SAAM system, where it goes first to the regional Server and then to the Server in the next higher level of the hierarchy.

Furthermore, the SAAM proxy is designed to support resident agents, allowing the Management Information Base in the router to upgrade dynamically. The precompiled byte code of a resident agent is registered as a new module by the hosting node. For instance, when the SAAM Proxy receives a server resident agent it becomes a SAAM Server, acquiring with this feature a flexibility that sometimes is necessary due to excessive traffic in the network or a failure of the regional Server.

For a thorough understanding of the SAAM system, it is necessary to address some aspects of Quality of Service characteristics in a network. Three main services establish the various QoS levels in a network: Best Effort Service, Integrated Service, and Differentiated Service.

(1) Best Effort. Best Effort Service is the basic model of the current Internet. It was the first service provided by the Internet. It gives no guarantees to clients such that they will receive sufficient bandwidth to support their offered load or that their packets will arrive at the destination within a specific time frame. Some applications such as real-time application would have problems with this lack of guarantees.

(2) Integrated Service. Unlike Best Effort Service, the Integrated Service provides a guarantee for a minimum bandwidth, as well as bounds on delay and packet loss rate. The client can negotiate for a specific QoS level based on the kind of application or session being supported. Guarantees are customized on a per-client

basis. The client application must go through the resource reservation process and wait for the service provider to establish a transmission path and reserve resources before it transmitting packets. This application denied the resource reservation if insufficient resources are available for handling the additional traffic.

(3) Differentiated Service. Due to the intense use of the Internet by real-time applications, the IETF first proposed an Integrated Services architecture, which uses an RSVP protocol to reserve network resources, to assure the service quality provided. The problem with this approach is the level of maintenance required by all the routers and end-systems to support a flow state. To address this maintenance issue, the Differentiated Service was created. The service is a mixture of the Best Effort and the envisioned per-flow heavyweight mechanism of RSVP and Integrated Service. The Differentiated Service accommodates heterogeneous application requirements and user expectations, permitting differentiated pricing of Internet Services and scalability.

(4) Example of Architecture. The figure below shows one possible test environment for the integration of service types. The configuration includes three Cisco routers. Linux boxes hosting the SAAM specific application code implement the SAAM system functionality. These boxes are called SAAM proxies. One SAAM Proxy is required for each Cisco router to be controlled by the SAAM system. Moreover, a proxy should be placed in the same local area network as its associated router so that the proxy is able to inspect the traffic going through the

router via a link-layer packet sniffer. In this test configuration, three SAAM Proxies are deployed. Each is directly connected to one router through the router's console port. The three proxies are also connected to the entire network by their network interface cards so that they can communicate with the routers and each other using the TCP/IP protocol.

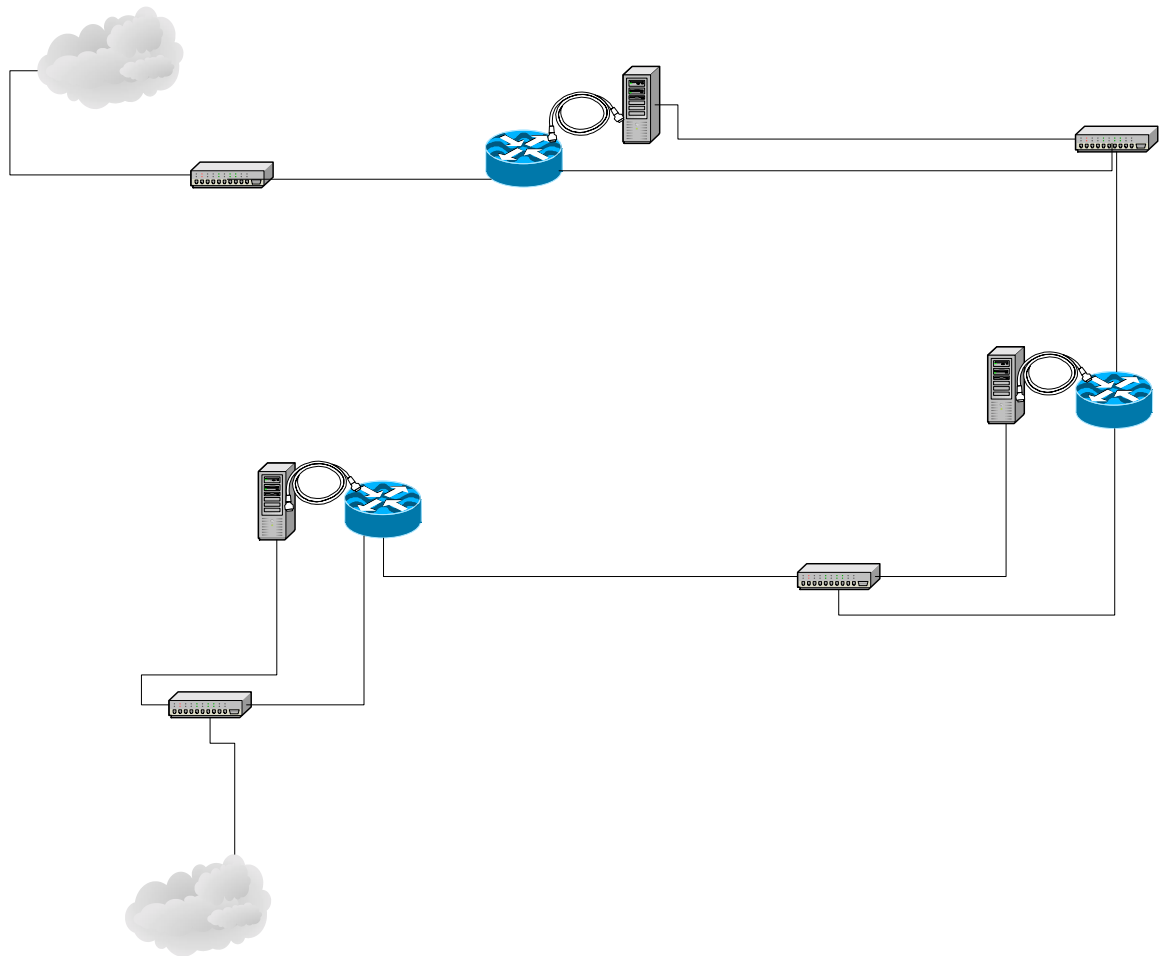


Figure 1. Example of a Network

C. CISCO ROUTER CONFIGURATION

1. IOS and IOS Command Line Interface

Cisco's IOS has many configuration values that establish the routing of information from one network to another. A software license is required in order to use the Cisco IOS.

Many versions of IOS exist. Deciding or choosing the particular version is based either on the need to implement a specific IOS feature or the desire to use a specific Cisco hardware platform.

Cisco uses a special numbering scheme to keep track of IOS versions. The full version number of an IOS has three numbers: major version, minor version, and maintenance release. The major version and minor version numbers are separated by a period and are referred to collectively as the major release. The maintenance release number is shown in parentheses. For example, the IOS version number 11.2(10) refers to maintenance release 10 of major release 11.2. Cisco releases updates often. When they issue an update for an IOS version, they generally increase a maintenance release number associated with the major number. Cisco also issues release notes that contain descriptions of release changes and additions.

Cisco uses special release designations to show how stable the software is. These release designations are as follows: General Deployment (GD), Limited Deployment (LD), and Early Deployment (ED). As a general rule, GD releases of an IOS are the most stable. Cisco puts the GD designation on an IOS release when it has been in the market long enough to have allowed Cisco to fix most serious bugs.

Feature sets do not change as often as version numbers. The selection is based on what is being run on the router. For example, the choice between Internet protocol (IP) or Novell's Internetwork Packet Exchange (IPX) needs to be done based on the desired features of the router and the required operation of the network.

Cisco has many models of routers that run a version of Cisco's IOS. They vary from inexpensive, low-end models to extremely expensive, high-end models. For the network backbone, the selection would be from among the high-end router series: 7000, 7200, 7500, or 12000. These series of routers are meant to be fast and reliable, and can support many network interfaces.

To connect an office LAN or WAN to the backbone, one of the access-type router series is appropriate: 1000, 1600, 2500, 2600, 3600, 3800, or 4000. The commands to configure a router's IOS are consistent across the entire IOS-based router line. This means that only one command-line interface must be learned. This interface looks the same whether talking to the router through a console port, a modem, or a telnet connection.

One of the major components of an IOS configuration is that of the individual network interfaces. Each Cisco router model requires network interface configuration. The possible Physical interfaces available on the Cisco routers include: Ethernet, Fast Ethernet, Token Ring, FDDI, Low-Speed Serial, Fast Serial, HSSI and ISDN BRI.

Most IOS configuration is done through the Command Line Interface (CLI). To configure the IOS-based routers, it is necessary to fully understand how the CLI works. The

change in the command prompt is based on the command mode that is being used on the mode in which the router is running. The major command modes are as follows: User Mode, Privileged Mode, Global Configuration Mode, Sub-Configuration Modes and ROM Monitor Mode. With the exception of ROM Monitor Mode, all modes are IOS command modes, used to configure the IOS.

When operating in the user mode, the prompt consists of the router's host name followed by the "greater-than" symbol (e.g. router2>). This mode cannot do anything that would affect the IOS operation. It only permits the user to see the IOS running configuration and startup configuration.

In the privileged mode, the IOS operation can be affected or its configuration can be shown. The user mode enable command tells the IOS that the user wishes to enter the privileged mode. When the enable command is entered, the IOS asks for a password before granting access to the privileged mode. This assumes an "enable" password has been previously configured and stored as part of the router's start-up configuration. The privileged mode is a superset of the user mode commands.

The command prompt for the privileged mode consists of router's name followed by a number or pound sign. At this point, the operator has complete control over the router configuration and operation. To return to the user mode, the command is disable.

The IOS configuration modes are used for entering IOS configuration commands that affect the way IOS runs on a router. The main configuration mode is the global

configuration mode from which several other configuration modes are entered. The configuration mode required depends on what is being configured and what command is being entered. All commands that are entered in a configuration mode affect the running configuration; these commands take effect immediately after they are entered. To make the changes persistent, the running configuration must be saved to the startup configuration, stored in nonvolatile RAM.

The configure terminal command is a privileged mode command used to enter the global configuration mode if a terminal is being used to enter configuration commands. IOS will accept the command only from the privileged mode.

The sub-configuration modes are used to configure individual components like interfaces and processes. The command to enter a sub-configuration mode varies based on the component being configured.

The ROM monitor mode is not really an IOS mode. It is rather a mode that a router can be in if IOS is not running. If a router attempts to boot and cannot find a good IOS image to run, the router will then enter, the ROM monitor mode.

D. THE CISCO ROUTER

Three different types of ports exist on CISCO routers with each having an associated connection. These connections provide the different ways of interfacing and configuring a router. Before determining how to configure the routers, one must answer the following questions:

- Can a "telnet" session be established with the console or the auxiliary ports of a router without having a LAN connection?

- Can the router's configuration be programmatically read and changed through the console port or the auxiliary port using a telnet session or a serial port communications program, like the "HyperACCESS" software from Hilgraeve? If so, how is it done? Specifically, what are the steps to be followed?

- If a router can be configured through its external ports instead of among the console, auxiliary and LAN connections, then which method achieves the fastest response time? How can the execution times of tasks, like reading a router's routing table or adding a static route, for example, be measured?

A Cisco router includes asynchronous serial console and auxiliary ports. These ports provide administrative access to the router either locally (with a console terminal) or remotely (with a modem). Thus, there are three methods of logging on to a router: Console port, Auxiliary port and a "telnet" session. Regardless of which connection method is used, access to the IOS command-line interface is generally referred to as an EXEC session and is accessed after entering the "enable" command at the user-mode prompt.

A router can also be accessed from a remote location by dialing directly into a modem connected to the Console or Auxiliary port of the router. In general, the console port is recommended because it displays router startup messages; whereas, the Auxiliary port does not provide this

information. In addition, if a router hangs in the read-only memory monitor mode, the system can be rebooted if the connection is through the console port. However, if a local terminal is connected to the remote router's console port, no other alternative may be available other to connect to the auxiliary port.

1. Cisco Console Port

The most basic way of logging into an IOS-based router is through a connection to the console port. The System Configuration Dialog asks for an enable password.

The console port is used as a direct attachment to a terminal. The IOS can be accessed through this using an RJ-45 port on the back of the router. The cable is wired such that the pin-outs are "complemented" at opposite ends: Pin 1 at one end connects to Pin 8 at the other; Pin 2 connects to Pin 7, and so forth. This configuration is referred to as a "roll-over" or "rolled" cable. It connects to an RJ-45-to-DB9 adapter, which connects to the terminal's serial port. The PC uses a terminal emulation program set to 9600 baud, 8 bits, N parity, and 1 stop bit. Another important aspect to know is that the console port has no password by default.

To connect to the console port, use a serial port communications program, like HyperACCESS or HyperTerminal, configured as 9600/8/N/1 and the "Flow Control" set to "None". The terminal emulation should be set to VT100. If the terminal is running Linux, the program, MiniCom, may be used. While Minicom has the ability to run scripts directly, other serial port applications require a custom

program to interface with the, communications application and to execute scripts through it.

a. *HyperTerminal*

HyperTerminal is a basic modem communications program that comes with the Microsoft Windows 95/98/Me/2000/XP operating systems. It is used as a mechanism to dial into an Internet Unix account to use text-based Internet programs, such as PINE, TIN and LYNX. HyperTerminal does not support PPP dial-up access; therefore, it cannot use graphical Internet Applications, such as Netscape, Eudora and Internet Explorer. A more recent version of HyperTerminal, the Private Edition 2.0, has fixed a few problems identified in the original version; and adds some useful features. HyperTerminal should be installed without explicitly making a request. The program that starts HyperTerminal is HYPERTRM.EXE, which is located under Start Menu: Programs: Accessories: HyperTerminal. In general, the use of HYPERTRM.EXE is necessary when adding a new phone number. If the location is saved after using HYPERTRM.EXE, the HyperTerminal directory will store unique icons for these different locations.

The HyperTerminal, like many Windows applications, has a button bar on the top of its screen that incorporates the most used features of the program. This application can transfer files between network devices, such as router to computer. It also allows the terminal to function as a telnet client.

b. HyperACCESS

HyperACCESS is the official upgraded application, built upon HyperTerminal and HyperTerminal Private Edition.

This application is a 100% 32-bit code and is built with the latest development tools to ensure full compatibility with Windows 95, 98, Me, NT, 2000 and XP. The application provides fast and reliable multi-tasking and supports long file names. It offers a wide array of additional capabilities, such as many additional terminal emulators and file transfer protocols; record and play back logons and repetitive steps, using Visual Basic Script and JavaScript; redefined keys and added buttons to the toolbar with text or bitmap labels; and automated communications with Visual Basic, VBA, C++, or other languages.

The HyperACCESS has a full support for Active X Scripting, OLE Automation, and the new Document Object Model for super integration with Microsoft Office applications and the Active Desktop. Additional capabilities may be developed using common high order programming languages, in conjunction with the API that improved the applications development. The Object Linking Embedding (OLE) Automation is used to take advantage of this ability.

HyperACCESS provides its own file transfer protocol, and HyperProtocol, which is faster than others, such as Compuserve B+, Kermit, Xmodem, lk-Xmodem, Ymodem, Ymodem-G, and Zmodem. These are some of the faster protocols available on the market.

Improvements to its display functionality permit users to expand the terminal view to occupy more of the

screen, and to prevent the image from following the cursor as it shifts out of view. It also includes new options to support host-controlled printing.

2. Remote Cisco Configuration

As with the console port, the auxiliary port can directly attach a terminal to the router. It is also useful for configuring a remote router, using a modem, from which the IOS can be accessed. Although Auxiliary port was originally created to support remote administrator access, many customers use it for dial-up back-up, particularly when analog lines are desired or when no others means is available. If a remote router stops responding, it can still be accessed when it has a modem on its auxiliary port. Additionally, it can be used to attach a terminal directly to the router for the purpose of sending packets from the terminal.

The Ethernet port permits the IOS to be accessed through the network via "Telnet" or "Secure Shell (SSH)" sessions. The "telephone network" as this communication is nicknamed, emulates a dumb terminal and connects over the network. Ethernet port can access the router from PCs or other routers on the network. When accessing the router, the user must be aware that "Telnet" sends and receives passwords and usernames in plain text, causing the implementation of the SSH on networks that need more security.

The flexibility of multiple port accesses means a router can be configured from many locations. Upon initial installation, the network administrator typically configures the networking devices from the console

terminal, which is connected via the console port. If the administrator is supporting remote devices, a local modem connection at the device's auxiliary port permits the administrator to configure those network devices.

After initial startup, there are additional external sources for software downloads accessible through router interfaces. Routers with established IP addresses that allow "Telnet" and "SSH" connections for configuration tasks can download a configuration file from a Trivial File Transfer Protocol (TFTP) server and can also be configured via a Hypertext Transfer Protocol (HTTP) browser. These methods assume an active IP configuration and network connectivity to the router.

With "Telnet" or "SSH", the connection to the router is through a LAN. If the router is already configured for the network it is using, one only needs to "telnet" to its IP address. The program, HyperACCESS, can also be used to "Telnet", so one can apply the same scripting, with very few changes, as done for the serial communications. With Linux, writing a shell script to use "Telnet" and execute the various required commands is necessary while sending all output to a file for later review.

a. Telnet

This application is a terminal emulation program for TCP/IP networks, such as the Internet. The Telnet program allows the host terminal to connect to a remote device and interact with that device as if it were directly connected to it. Possible devices, which may be accessed using Telnet, include both servers and routers along with other appropriately configured workstations. Telnet allows

a terminal, such as a PC, to access remote, Telnet capable devices over TCP connections and to execute console commands. The Telnet server can pass data it has received from the client to many other types of devices, including a remote login server.

Telnet uses the Network Virtual Terminal (NVT), a set of communication facilities that utilizes TCP/IP protocol. At the user or client end, the telnet client program is responsible for mapping incoming NVT codes to the actual codes needed to operate the user's display device. It is also responsible for mapping user generated keyboard sequences into NVT sequences.

The NVT uses seven-bit codes of characters and is only required to display the standard printing ASCII characters represented by those seven-bit codes. It must also recognize and process certain control codes. The seven-bit characters are transmitted as eight-bit bytes, with the most significant bit set to zero. An end-of-line is transmitted as the character sequence, carriage return (CR) followed by a line feed (LF).

A variety of options can be negotiated between a telnet client and a server using commands at any stage during the connection. They are described in detail in separate RFCs.

Options are agreed upon by a process of negotiation, which results in a client and server having a common view of various extra capabilities that affect the session and the operation of applications. Sub-option negotiation allows some of the negotiable options values to

be communicated once both devices have agreed upon support of a particular option.

Further information can be found in RFC854, which was first published in 1983.

b. Perl

The Practical Extraction and Report Language (Perl) optimized for string manipulation, I/O, system tasks, and incorporate syntax elements from the Bourne shell, csh, awk, sed, grep, and C. It provides a straightforward and powerful interface to TCP/IP, while making it possible to create robust, maintainable, and efficient custom client/server applications. Additionally Perl is widely used in the World Wide Web as a quick and effective way to model applications that provide much of the web's interactivity.

Larry Wall established this language and first posted Perl in 1987. He created it as a text processing language for Unix-like operating systems. Version 4 of Perl had become very stable and was a standard Unix programming language. It worked well when was used in small programs; however, this version was heavy in large software applications. The first release of version 5 came out in late 1994, making this language a convenient tool for system administrator. Based on natural language, like English, it is an easy way to write programs, providing flexibility and clarity. Some of its modules are very useful, such as support for Telnet, Net::Telnet. It includes some additional functionality for dealing with Cisco routers Net::Telnet::Cisco. These modules were

developed as free software and can be redistributed and modified under the same terms as Perl itself.

This chapter began with an assertion regarding the necessity of dynamically managing network resources autonomously to satisfy the various constraints of QoS guarantees. It then provided a short description of several of the many tools available for monitoring and controlling network resource usage. The next chapter will provide an example of custom application software used for communication between a remote terminal and another computer or a Cisco router. This application is a good demonstration of the utility of the Perl scripting language.

THIS PAGE INTENTIONALLY LEFT BLANK

III. COMMUNICATION USING PERL SCRIPT

A. SIMPLE NETWORK MANAGEMENT PROTOCOL

To achieve the goal of this research, the proxy must communicate with its associated routers and other proxy servers in a timely and secure manner.

Uploading the proxy is specific QoS parameters, such as flow definitions and routing table entries, to a Cisco router and extracting current state information from the router requires software providing communication that is fast, efficient and suited to the task of the remote router configuration management.

If there are control mechanisms in the router, such as RSVP, Integrated Services, and Differentiated Services, may be easier integrating capabilities of a proxy to the network. Prior studies attempted a solution using the Simple Network Management Protocol (SNMP).

The SNMP was created for providing remote control of many different types of network devices. It easily provides the ability to remotely monitor, configure and receive notification of important or useful events from the routers. Many network management tools are based on SNMP for just that reason.

In the Cisco environment, the router will contain an SNMP agent and generate the Management Information Base (MIB) used by the agent. An MIB is information that represents parameters of the network device, such as traffic, CPU processes, management information, etc. These values are integers, strings, counters, and others. A manager can request a value from or store a value into a

particular MIB. An agent is responsible for collecting and maintaining information stored in the MIBs about its local environment. The agent provides the information to a manager, either in response to a request or in an unsolicited fashion when something noteworthy happens.

Previous studies have concluded that it is impossible to use the SNMP to upload some of the SAAM specific configurations into a router because such capabilities would require modification of the vendor's proprietary operating system, and would need the vendor's support. However, the protocol may be a better choice for extracting router state information than the IOS CLI because the latter option requires parsing of complex command output text strings.

Therefore, finding another mechanism capable of controlling the router is necessary. Several ideas have been considered, such as using an agent to translate the proxy directives into commands understandable by the router's operating system and then using the router's standard interface to dynamically configure the router.

However, maintaining quality of service guarantees requires the router status and performance be continuously monitored. In the same way, the proxies need the capability of dynamically interacting with each other.

Thus, establishing automated interfaces to the router and to the proxies, using scripting languages, is essential to the task of remotely configuring, monitoring, and controlling the proprietary network traffic delivery devices.

The Perl script language is not only appropriate but also fast, efficient, and easy to learn. It is one of the scripting languages used in this research. In the last chapter Perl was introduced as being an interpreted language that is optimized for string manipulation and as being a programming language that uses a natural language for writing code.

1. Configuration Experiment

First of all, reiterating the goal of this project is necessary. The goal is to develop an effective way of communicating between proprietary devices and computers. Understandably the research did not focus on all of the commands for controlling the QoS in the network, however, the research provides insight just enough into the ability to update and control a router through a software application.

The main topology used for this research included a notebook and a desktop computer with a direct link to a hub. This hub was linked to one of the Ethernet ports of a CISCO router. A desktop computer was also linked through its serial port to the console port of the router. The laptop computer was connected to a printer through its serial port. The topology is diagrammed in the figure below.

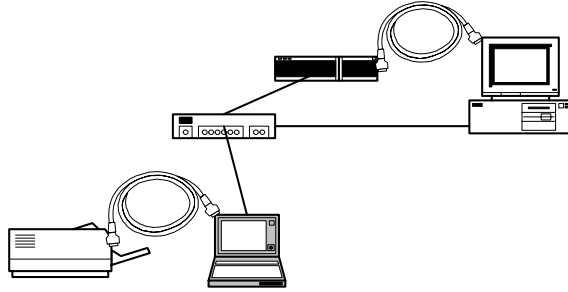


Figure 2. Layout of the Demonstration Network

To work with Perl scripts some software needs to be installed, such as Active 5.6.1, which is the Perl program that includes the module packets, the interpreter software, and the Open Perl IDE, This IDE is a user interface for writing and debugging Perl scripts for use with Windows 95/98/NT/2000.

Depending on the purpose of the Perl script being developed, some packages may need to be installed to increase the utility of this Perl IDE. In the case of this project, the modules `Net::Telnet::Cisco` and `Net::SSH::Cisco` were included to provide communication using "Telnet" and "SSH" with the Cisco router.

The `Net::Telnet` module provides utilities to support client connections to a TCP port using the Telnet protocol. Telnet is a character-oriented application. Simple methods, such as `print` and `get`, or many more sophisticated features are provided since connecting to a Telnet port was originally intended for human interaction with the remote or host system. The module includes the ability to specify a time-out parameter and to wait for pre-specified character patterns to appear in the input stream, such as the prompt from a shell. This package provides a simple

way to make client connections to TCP services. These services may include the means to specify the connection time-out period, reading or writing files or configuration parameters and may communicate with an interactive program through some sockets or pipes waiting for certain patterns to occur.

Net::Telnet::Cisco provides additional functionality to Net::Telnet specific to working with Cisco routers. Some files, such as cisco.pm, need to be appended to the directory, Net/Telnet/Cisco, before having the desired effect in this module's use. The directory is created when the module is installed

Another approach for connecting through a Transmission Control Protocol (TCP) session is the use of the Secure Shell (SSH). This remote connection application provides enhanced security over Telnet. Note that Version 1 is the only one implemented in the Cisco IOS software.

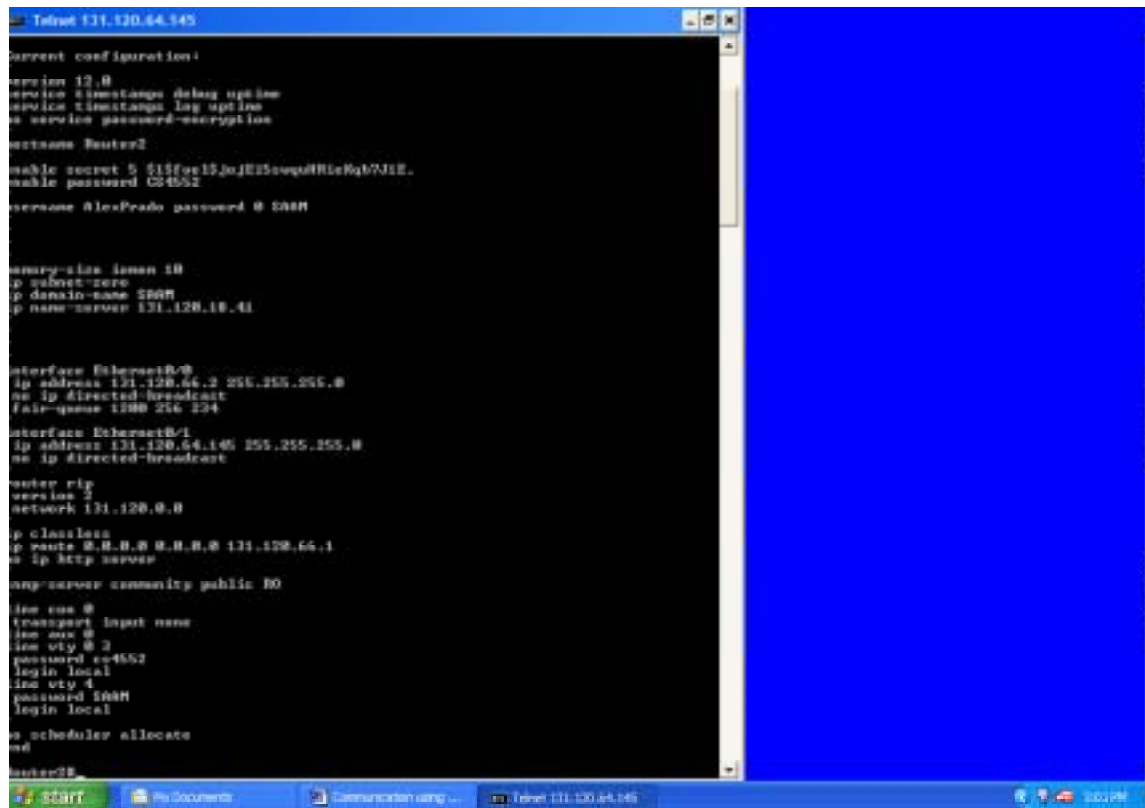
The SSH server feature enables a SSH client to make a secure, encrypted connection to a Cisco router. The server will work with publicly and commercially available SSH clients. It supports DES (56 bit) data encryption and Triple DES (168 bit) data encryption software images.

The authentication, authorization, and accounting (AAA) feature and the IP Security Protocol (IPSec) feature are available in the Cisco routers. However, the Cisco IOS security Configuration Guide, Release 12.1, and Cisco's IOS Security Command Reference, Release 12.1, specify that the router must have an IPSec encryption software image from Cisco IOS Release 12.1(1) T installed on the router.

a. Uploading a Router Configuration Using Perl Script

The main purpose of a configuration upload capability is to maintain a dynamic and timely configuration of the router to support the QoS management without the network suffering from any problems due to the configuration traffic.

The lab topology described above will be used to measure the time necessary for the commands to upload to the Cisco router. Some screenshots were captured to verify that the configuration was uploaded and implemented successfully. These screens include the configuration of the router before using telnet access (Figure 2), and the Open Perl IDE showing the result of the upload session, including the elapsed time and the commands executed (Figure 3).



```
Telnet 131.120.64.145
current configuration
version 12.0
service timestamps debug uptime
service timestamps log uptime
! service password-encryption
hostname Router2
enable secret 5 $1$fe1$joJEISoqo0R1eNyb7NIE.
enable password C04552
version AlecPrado password 0 SNN

memory-size 16m 10
ip subnet-zero
ip domain-name SNN
ip name server 131.120.18.41

interface Ethernet0/0
ip address 131.120.64.2 255.255.255.0
no ip directed-broadcast
fair-queue 1280 256 124
interface Ethernet0/1
ip address 131.120.64.145 255.255.255.0
no ip directed-broadcast

router rip
version 2
network 131.120.0.0

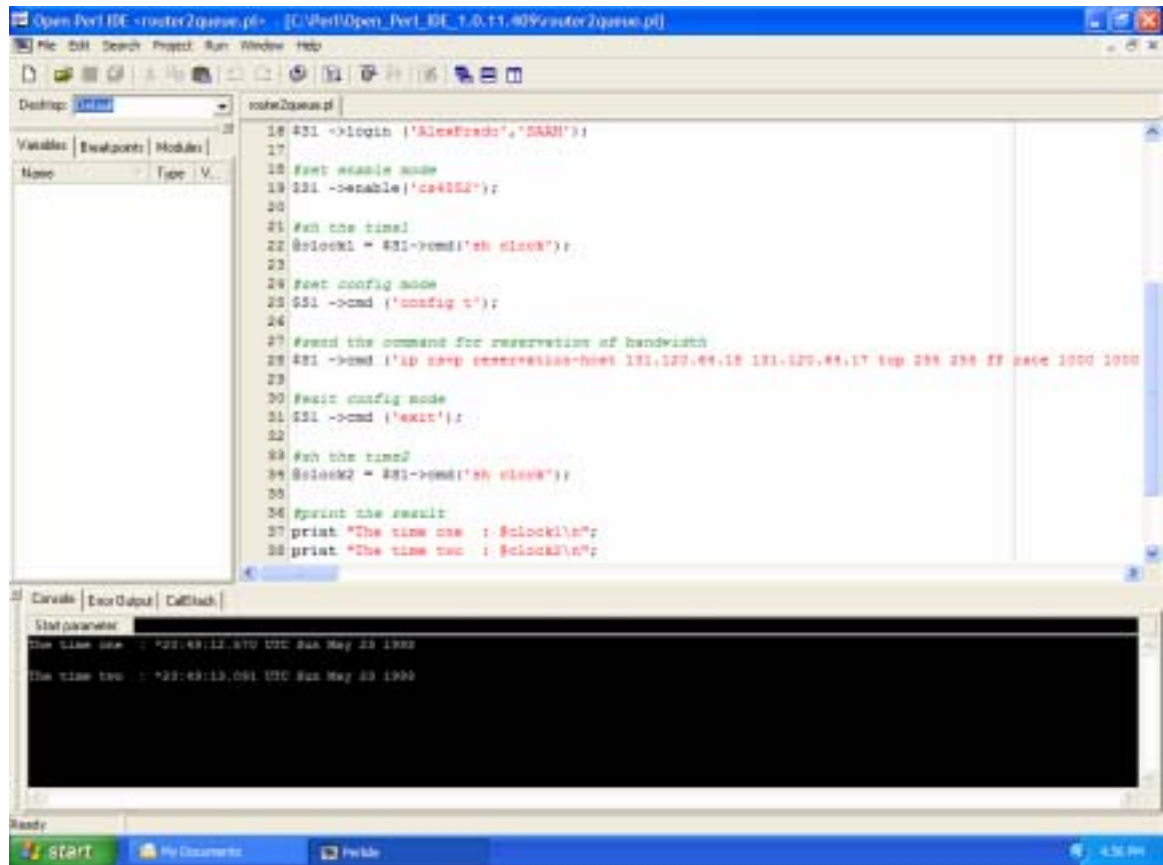
ip classless
ip route 0.0.0.0 0.0.0.0 131.120.64.1
ip http server
http-server community public 80
line con 0
transport input none
line aux 0
line vty 0 3
password c-4552
login local
line vty 4
password SNN
login local
! scheduler allocate
end
Router2#
```

Figure 3. Configuration of the Router Using Telnet

Figure 3 shows the script of the Perl program containing the Cisco IOS command `ip rsvp reservation-host 131.120.64.18 131.120.64.17 tcp 255 255 ff rate 1000 1000`. This script configure a static RSVP interface, with a reservation established between the destination hosts, 131.120.64.18 and 131.120.64.17, using TCP port 255 for both destination and source interfaces. It establishes a single reservation (ff) with a guaranteed bit rate service (rate) of 1000 kbps and a reserved average bit rate of 1000 kbps.

The difference between the start and completion times, which were retrieved using the command, `sh clock`, and stored in the variables, `@clock1` and `@clock2`, is 521

ms. This proves that this kind of communication can quickly upload the desired configuration to the router.



The screenshot displays the Open Perl IDE interface. The main editor window shows a Perl script named 'router2queue.pl'. The script includes comments in green and code in black. It starts with a login command, enters enable mode, sets a clock, enters configuration mode, and issues a bandwidth reservation command. It then sets another clock and prints the time before and after the command. The output window at the bottom shows the execution results, including the time taken for the command.

```
16 $SI ->login ('AlicePado',"DAM")
17
18 #set enable mode
19 $SI ->enable ('ca6662');
20
21 #set the time1
22 $clock1 = $SI->cmd('sh clock');
23
24 #set config mode
25 $SI ->cmd ('config t');
26
27 #send the command for reservation of bandwidth
28 $SI ->cmd ('ip rdp static-queue-hoat 131.127.69.18 131.127.69.17 top 256 256 27 page 1000 1000
29
30 #set config mode
31 $SI ->cmd ('exit');
32
33 #set the time2
34 $clock2 = $SI->cmd('sh clock');
35
36 #print the result
37 print "The time one : $clock1\n";
38 print "The time two : $clock2\n";
```

Std/Output

```
The time one : *23:48:12.870 UTC Sun May 23 1999
The time two : *23:48:13.081 UTC Sun May 23 1999
```

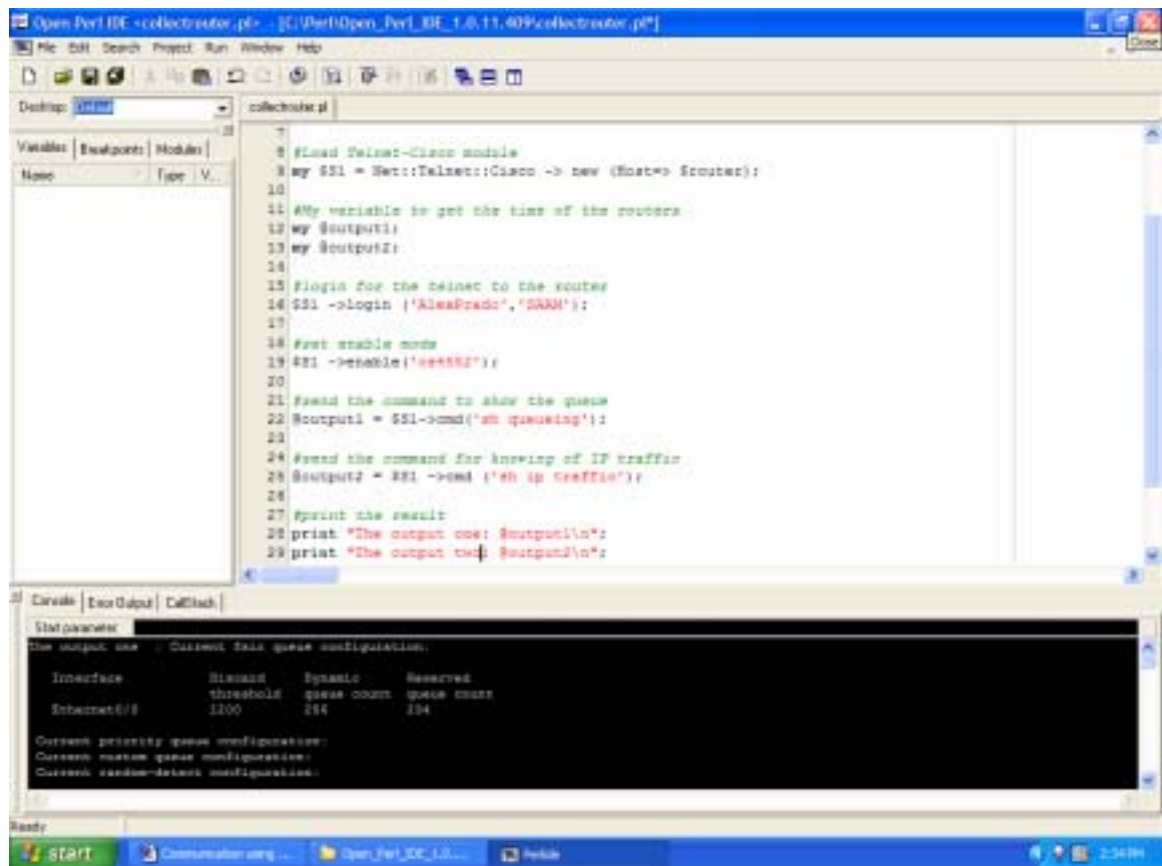
Figure 4. Upload data with the Open Perl IDE

b. Collection the Router Configuration Using Perl Script

As noted earlier, collecting the data on bandwidth, reliability, and interface packets counts, and other crucial information on the health of the network is crucial for maintaining a base of all of the proposed services in the network. Based on the same approach used in the experiment where the router was updated using a Perl script, the screenshot below shows some essential information taken from the router. This information is available by using some Cisco commands and comparing them

to the parameters settings necessary for maintaining the monitored network's required QoS level.

Using the show queueing and show ip traffic commands, the script acquires the state of the queue in the router and the state of the router's traffic. It is important to mention that almost all the commands for collecting the data from the Cisco router are executed in the Privileged Mode of the IOS Command Line Interface.



```
1 #Load Telnet-Class module
2 my $SI = Net::Telnet::Class -> new (Host=> $router);
3
4 #My variable to get the time of the routers
5 my $output1;
6 my $output2;
7
8 #login for the telnet to the router
9 $SI ->login ('AlmaPrado','DAM');
10
11 #set stable mode
12 $SI ->setmode('setmode');
13
14 #send the command to show the queue
15 $output1 = $SI->cmd('sh queueing');
16
17 #send the command for knowing of IP traffic
18 $output2 = $SI ->cmd ('sh ip traffic');
19
20 #print the result
21 print "The output one: $output1\n";
22 print "The output two: $output2\n";
```

StdOut:

```
The output one : Current fair queue configuration:
Interface      Bandwidth  Dynamic    Reserved
threshold      queue count queue count
Ethernet0/0     1000      256       256

Current priority queue configuration:
Current mls queue configuration:
Current random-detect configuration:
```

Figure 5. Collection of the data with Open Perl IDE

c. Automatic and Timely Communication Between Different Managing Agents

The Perl script was used for communication between the proxies, based on the idea that each proxy agent must communicate with other proxies in a timely and secure manner. Socket mechanisms were used for this kind of link. They permit the program to communicate, either within the same machine or across the network, where each entity is identified by a unique address.

The employed method creates a program for receiving a connection request, and then asks the operation system to create a connection and bind it to a port. The program listens to the socket created for incoming connections. The same approach is used for creating an outgoing socket for communication with the caller. The caller needs to specify an address and a port number for the receiving end.

After the two sockets are configured they can exchange information, each by writing to and reading from the associated sockets.

Perl provides support for the socket API through the module, `IO::Socket`. The API works with the script to create a wrapper for the Cisco commands.

The figure below shows one code example that was created to receive a message from another host. The expected message was "Communication with SAAM Proxy." The code is used to receive all the connections interested in exchanging information.

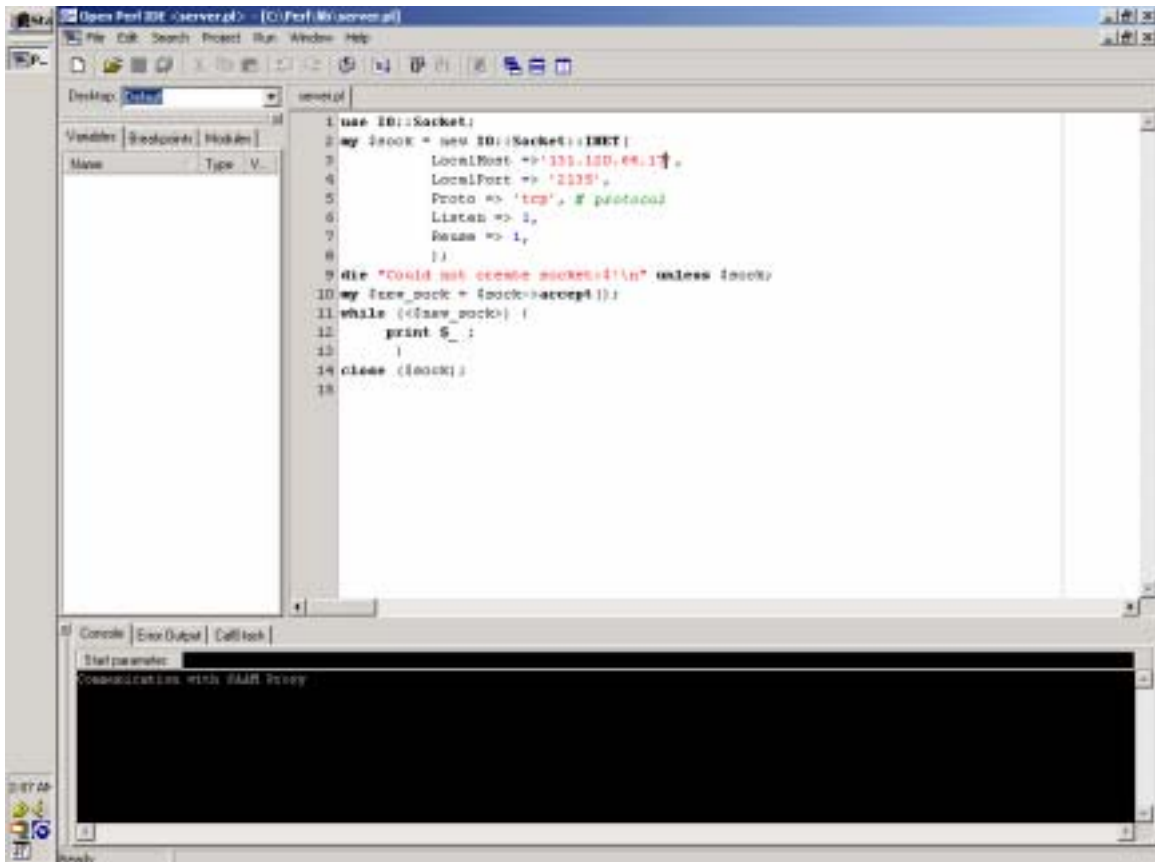


Figure 6. Receipt of a Message from Another Socket

The other participant in the communication needs to create another socket with the address and port of the station to which the host is interested in sending information. In addition the receiving station needs to be prepared by creating the required socket. To do so, the code that creates the socket must be executing.

Another important issue is that the communication can reach a deadlock if the two sockets try to read or write simultaneously and no a way exist to determine whether or not the other end has finished sending data. There must be a protocol between them that denotes logical sections of the communication in the contents of

transmitted messages. Therefore, the two sockets must follow a common procedure, upon which they have agreed, when exchanging data.

Following the example above, the figure below shows a code with the embedded message, "Communication with SAAM Proxy," which was sent.

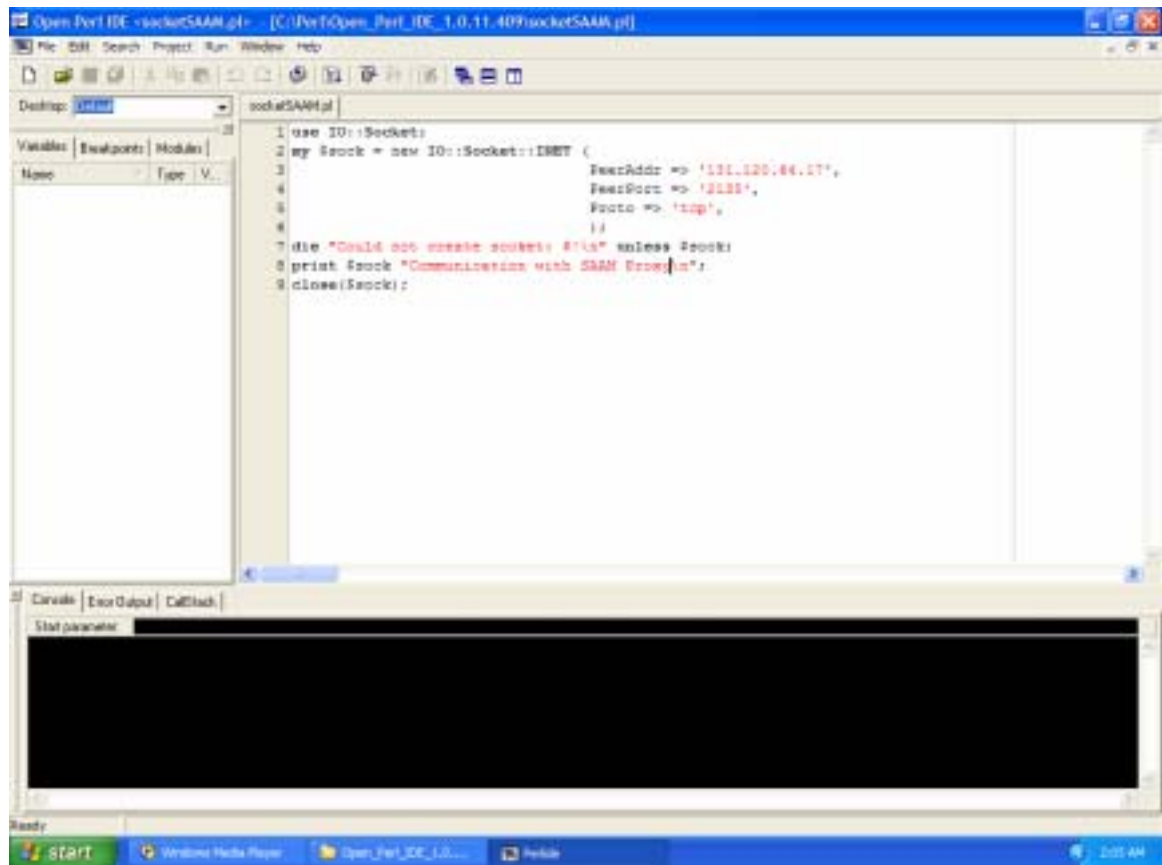


Figure 7. Transport of a String to Another Socket

d. Code and Answers

There are a lot of possibilities are available for using for using Perl scripting to satisfy the intent of this thesis. The project included one example of each kind of required communication in order to give a good idea what Perl can do in the context of router configuration control.

The presented programs demonstrated how to upload and to collect data from the Cisco routers. They also showed how to send information between two agents in the network. Further information on the use of Perl can be found in two ways: either by exploring Internet sites, such as www.perldoc.com or in books documenting the Perl language. For the purpose of this project, either the commands or modules relevant to network communications are essential in understanding how to configure them.

Knowing about the Cisco commands is critical for manipulating the IOS Command Interface to obtain the desired information. Such manipulation can extract a great deal of important data. Surprisingly, manipulating the commands in the Cisco router can improve the network performance. Again, the command information can be learned either from Internet sites, such as www.cisco.com, or in Cisco publications.

The Perl code for uploading and downloading the router configuration setting are shown in the appendices, along with some of the results observed during its execution. However, a short explanation of that code follows to facilitate understanding.

(1) Upload and Collection of Cisco router data. The following Perl statement shows how to import or integrated modules in the code:

```
use strict and use Net::Telnet::Cisco.
```

The next two lines define the address where the connection was made, in case, with a Cisco router and was assigned it to a variable.

```
my $router = '131.120.64.145'
```

```
my $S1 = Net::Telnet::Cisco -> new (Host=>
$router).
```

The variables used to store the time information included

```
my @clock1, my @clock2, my @output1, my
@output2.
```

The login information for establishing the Telnet session with the username and the password, was sent to the socket and \$S1, by the following statement:

```
$S1 ->login ('AlexPrado','SAAM').
```

Likewise, the "enable" command and the privileged mode password was sent to the router by the following statement:

```
$S1 ->enable('cs4552').
```

The configuration upload start time was extracted from the router with the following statement:

```
@clock1 = $S1->cmd('sh clock').
```

Router queue status was retrieved and loaded in the variable, @output1, by the statement

```
@output1 = $S1->cmd('sh queueing').
```

Router traffic information retrieved and stored by the command

```
@output2=$S1->cmd('sh ip traffic').
```

The router's configuration mode was set by

```
$S1 ->cmd ('config t').
```

The statement to send the command to set the reservation for the path in the router was

```
$S1 ->cmd ('ip rsvp reservation-host
131.120.64.18 131.120.64.17 tcp 255 255 ff rate 1000
1000').
```

To print the clock results, the following statement was executed:

```
print "The time one : @clock1\n".
```

Finally, to exit the command mode and terminate the telnet session, the following statements were executed.

```
$S1->disable and $S1->close.
```

(2) Communication with two managing agents. The module described below provides an easier way to manage the socket using a native API wrapper:

```
use IO::Socket
```

This command creates a socket specifying the parameters need for creation.

```
my $sock = new IO::Socket::INET (
```

Local Host=> specified the local hostname.

Local Port=> defined the local port to communicate.

PeerAddr=> specified the remote address for the information.

PeerPort=> specified the remote port for the information.

Proto=> specified the communication protocol to be used.

Listen=> established the maximum number of connections that can be queue to the socket.

Reuse=> directed the system to reuse the port after the program exits.

The method accept() returns a new socket that can communicate with the requesting socket.

```
my $new_socket = $sock ->accept ().
```

The information that arrived was read or written to the new socket and was then displayed on the

screen. This was done in an infinite while loop which terminates when the socket is closed

```
while (new_socket){  
    print $_  
}
```

Error conditions, encountered either during socket creation or communication, were displayed on the console screen using the enclosed string.

```
die "Could not create socket: $_\n" unless  
$sock.
```

The information extracted from the socket was displayed with the following statement.

```
print $sock "Communication with SAAM  
Proxy\n".
```

Finally, the following statement was used to close the connection.

```
close($sock).
```

B. CONCLUSION

This chapter provided an example of how Perl scripting can be used to monitor, manage, and control the configuration of Cisco routers. It also described a network topology set up to experiment with Perl script based remote router configuration. The chapter then examined the semantics of the sample scripts.

The Perl method requires Telnet or a similar TCP/IP based protocol to remotely access the Cisco IOS CLI. The next chapter describes how to access the IOS CLI directly via the serial interface of the router's console port. An application program performs this direct access

written in C++ or Visual Basic and is made of building blocks provided by a serial communication package called HyperACCESS.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. ROUTER CONFIGURATION USING CONSOLE PORT

A. HYPERACCESS

The most direct way of logging into an IOS-based router is by connecting to the router's console port. The console port based access is desirable because it is much easier to secure a console connection than a dial-up connection or a telnet connection. However, the console port is designed for manual, interactive communication with the router. Therefore, a software utility must be developed to automate the console port access.

After initial research, it was concluded that HyperACCESS, developed by Hilgraeve, is a good software platform for automating communication between a control agent and its router using the console port. HyperACCESS provides an Application Programming Interface (API) that can be called from JavaScript or VBScript programs. The API may also be integrated with code written in full-fledged programming languages, such as Visual Basic (VB) and C++. The VB or C++ support is important because the resulting code is more efficient and more extensible than high level scripts.

This research makes use of HyperACCESS Version 8.3. HyperACCESS provides some tools for automating interactive and often repetitive tasks, such as production and delivering of many keystrokes to the remote system and waiting for prompts from a remote system before sending responses. These tools are used to execute sample VBScript or JavaScript programs to communicate with the router

through a serial link. Several VB and C++ examples are also developed to demonstrate how to use the low level HYPERACCESS libraries.

B. SCRIPT LANGUAGES

The HyperACCESS package provides access to the HyperACCESS API (HAPI) by any Object Link Embedding (OLE) Automation enabled with external a programming language. HyperACCESS has a GUI based programming tool for VBScript and JavaScript generation. Writing, testing, and running programs is possible without leaving the GUI application. Alternatively, scripts and programs can be developed with external development systems and integrated through HAPI. In this work, many VBScript or JavaScript scripts were created to test the hypothesis that router configurations can be autonomously queried and updated to satisfy QoS requirements. Most of these were not automatically created using the HyperACCESS tool because uploading configurations to and collecting data from the router require many specific calls to HAPI functions.

A program that uses ActiveX scripting (either VBScript or JavaScript) has a file name with a .txt extension. When the program is used as a function for another language, such as C++ or Visual Basic, it is compiled with the program using that language. This produces a regular executable file with an .exe extension.

1. Configuration Experiments

The experiment topology is the same as shown in Figure 1 in Chapter III. To establish a serial connection with the

router's console port, the HYPERACCESS terminal emulation program was configured to use the COM 1 or COM 2 port of the host computer with the connection parameters set to 9600 baud, 8 bits, N parity, and 1 stop bit. After the serial connection was established with the router's CLI, launching command scripts from within the terminal emulation program configured the router. The same setup was used for testing each JavaScript or VBScript program. Experiments were also conducted to establish and utilize agent-router serial connections with customized programs written in C++ or Visual Basic, which are linked with the HyperACCESS libraries.

The experiments first evaluated programs in JavaScript or VBScript. The examples demonstrate that it is possible to upload a configuration to a router or collect data from it using VBScript and JavaScript programs through a HyperACCESS connection. Such actions can be very useful for communicating between the SAAM server and its routers.

a. Upload of a Router Configuration Using HyperACCESS with VBScript

As noted, HyperACCESS has an Applications Programming Interface (API) that can be used by a VBScript or JavaScript. It is a good tool for effectively maintaining a dynamic router configuration in a responsive manner to support the QoS requirements of the network, using the console port of the router.

In the experiment, a VBScript object was created using the HAPI to upload a router configuration command. Based on the same approach used for Perl scripts, where the necessary time to enter and execute the command was

measured to assess the utility of the upload, the VBScript enters and executes a configuration command while measuring the elapsed time.

Figure 8 shows the HyperACCESS display of the interchange between the console and the router as the RSVP configuration command, `ip rsvp sender 131.120.64.145 131.120.64.18 tcp 0 0 131.120.66.2 Ethernet0/1 3750 7500`, is entered and executed. This command configures a static RSVP path between the destination address, 131.120.64.145, at Port 0, and the source address 131.120.64.18, on Port 0. The transport protocol, TCP, is specified, and a previous hop address of 131.120.66.2 is configured while an average data rate of 3750 kilobytes is reserved and a maximum burst rate of 7500 kilobytes is declared. While the router due to the lab's restricted configuration rejected the command, the experiment successfully demonstrates the ability, using a VBScript, to send a command to a router through the console port and cause it to execute.

(1) The code for a command upload. The VBScript technique was chosen for demonstrating the upload of a configuration command to the router. Notepad was used as the text editor, resulting in a file with a ".txt" extension, as noted above. All the commands below are in bold letters and the result of their execution is shown in Figure 7. The key script statements are preceded by a short description.

The language declaration statement specifies to the HAPI which scripting or programming language was used to define the application attempting to establish a

HyperACCESS link. Note that the name of the router to be commanded is "Router2."

```
$LANG = "VBScript"
```

The variable, `cr`, is initialized to the value of the carriage return character before the subroutine declaration, giving `cr` a global scope. This allows it to be used in event subroutines without being declared again. This variable is used with string constants to send the router text strings it would normally be receiving from an interactive user at a keyboard. Since the console port utility normally interacts with a human on a keyboard, the carriage return indicates to the router that a command has been completed and should be acted upon.

```
cr = Chr(13)
```

Should an error occur with any HAPI function the connection automatically aborts the script, based on the following command.

```
haAbortOnError
```

The global variable, `cr`, is transmitted as a string of characters to the console port. Initialized to a carriage return, the string represents the initial carriage return that the router is expecting to initiate as a session with a console terminal user. Thus, the following command enables the link between the HyperACCESS application and the router.

```
haTypeText cr
```

Since the console port configuration facility of the router was designed as an interactive, text-based session, delays must be inserted in the script to allow the router to properly respond before another command is entered. The specified function causes the script to pause until the string, "Router2>," is received

from the router. This is the normal response by the router to an initial carriage return entered either through a console port or a Telnet session. If this string does not appear after 100 seconds or it is not followed by a .3 second delay the program times out and exits.

```
haWaitForPrompt "Router2>"
```

The command "en" and the carriage return are sent to the router as a string of characters. When received by the router, the string is interpreted as a command to go to the privilege mode of the router.

```
haTypeText "en"&cr
```

The script must then wait for the string "ssword:" (password), which is querying the console for the privilege mode password.

```
haWaitForPrompt "ssword: "
```

The password used for the lab is "cs4552" to which the carriage return appended and the resulting string transmitted.

```
haTypeText "cs4552"&cr
```

As before, the script must pause until the router sends the appropriate response, in this case the string "Router2#," which indicates the router has entered the desired privileged mode.

```
haWaitForPrompt "Router2#"
```

The command "sh clock" and the carriage return are then sent as a string of characters, asking for the router's exact time to be returned to the script.

```
haTypeText "sh clock"&cr
```

This returned time value is then loaded in the variable t1.

```
t1 = Timer()
```

The command "config t" and the carriage return are sent to the router directing it to enter the "configure from terminal" mode

```
haTypeText "config t"&cr
```

The script must then wait for the router to respond with the string, "config)#," which indicates the router has entered the mode allowing the configuration to be modified through the console. The entire string returned by the router is Router2 (config)#.

```
haWaitForPrompt "config)#"
```

At this point the script can configure a static RSVP path, as described above. Note that the carriage return variable is appended to the string constant and the resulting string is sent to the router.

```
haTypeText "ip rsvp sender 131.120.64.145  
192.120.64.18 TCP 0 0 192.120.66.2 Ethernet0/0 3750  
7500"&cr
```

The command "exit" is entered in response to the configuration mode prompt, causing the router to return to the privilege mode.

```
haTypeText "exit"&cr
```

The router's exact time is again extracted and the result assigned to the variable, t2.

```
haTypeText "sh clock"&cr
```

```
t2 = Timer()
```

The command "exit" is then sent to close the console port interface

```
haTypeText "exit"&cr
```

The difference between the time value stored in t1 and t2 is then computed to determine the elapsed time.

```
t3 = t2 -t1
```

A dialog box, with an appropriate message, is displayed to the user on the terminal screen.

```
haMessageBox "Difference of time", t3, 1
```

Finally, the script ends the HyperACCESS session.

```
haTerminate
```

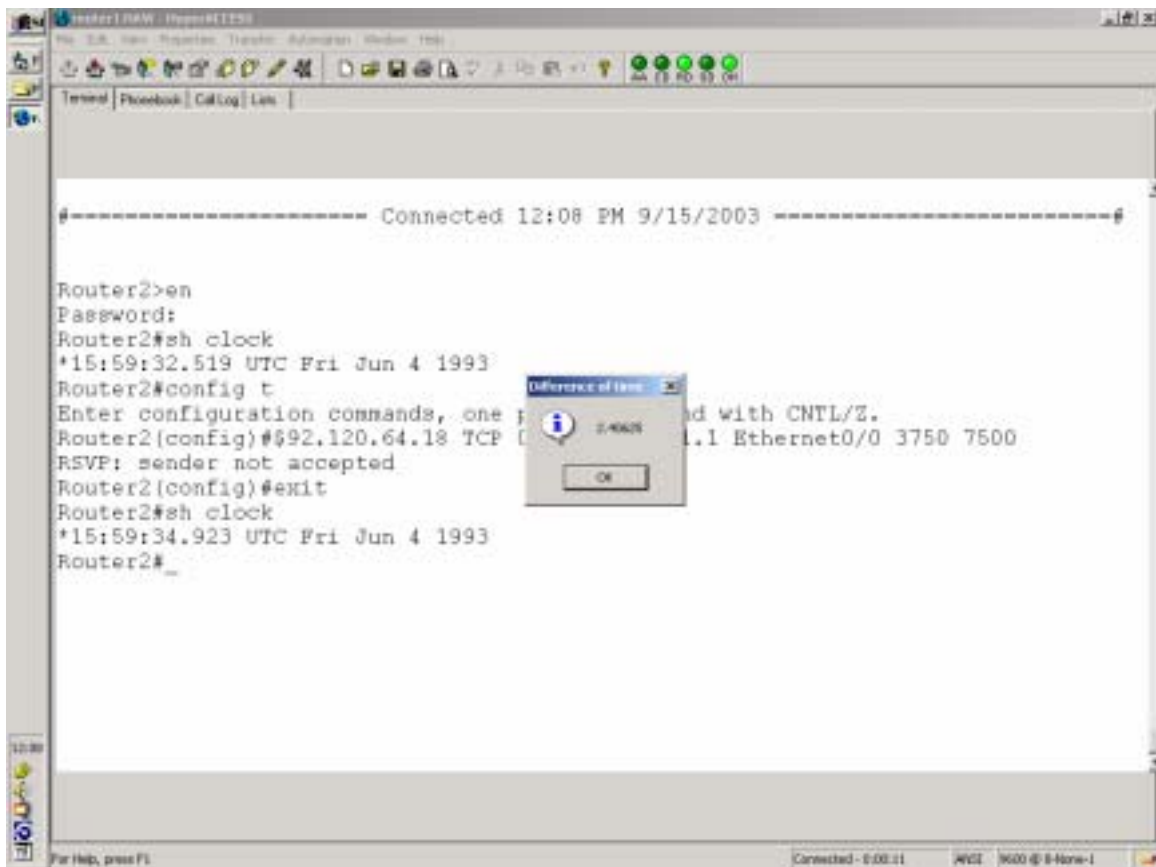


Figure 8. HyperACCESS Linked to the Router Showing the Result of the Elapsed Time for the Configuration Update Interaction Using a VBScript Application

The elapsed time, 2.40625 seconds, was not as short as that experienced when using a Perl script and Telnet. The main reason is that the software HyperACCESS interacts with the router, command by command. Therefore, it requires a delay after each command line while waiting for the resulting response from the router.

There is also a way to hide the HyperACCESS program from the viewer, using the API "haSizeHyperACCESS" with parameters HA_S_HIDE. The function is normally used in a script created object using a language of programming such as Visual Basic or C++. This allows the application to run in the background without delaying for updates to the console terminal monitor.

b. Collection of Router Configuration Information Using HyperACCESS with JavaScript

Information collected from the router's running configuration is essential for maintaining the network's QoS level. Extracting the router's interface status was based on the same approach as that used when updating the router's configuration using HyperACCESS. The key difference is that the command sequence, shown in the Figure 8, does not measure the required time to update a router configuration parameter, but rather extracts the interface queue status. Furthermore, the script used was JavaScript, demonstrating the flexibility of the HyperACCESS utility.

(1) The code for collecting information from the router. Following the same approach used above to describe the key statements of the VBScript, the bold words below are the JavaScript commands. Among the differences are that the JavaScript that uses parentheses to encapsulate all the function call parameters and that complete each command with a semicolon.

The language with which the HAPI was accessed is first declared, as in the case with the VBScript, above.

```
$LANG = "JavaScript"
```

The variables, vtr and cr, are declared, and the latter initialized to the ASCII value of a carriage return

```
var vtr;  
var cr = "\r";
```

As with the VBScript, an application termination is forced if an error occurs.

```
haAbortOnError();
```

A carriage return is sent to the router to initiate the console session. The carriage return, in this case, is the single character string, cr, initiated above.

```
haTypeText (cr);
```

As noted in the previous example, the script must wait for the proper response to be returned by the router to each interactive command. If this expected string does not appear after 100 seconds or is not followed by a .3 second delay the program times out and exits. In each delay statement the key phrase is included as a string and specified to what value the function is to react.

```
haWaitForPrompt ("Router2>");
```

The command "en" and the carriage return are transmitted as a string to direct the router to enter the privileged mode.

```
haTypeText ("en"&cr);
```

The script then waits for receipt of the password prompt string.

```
haWaitForPrompt ("ssword: ");
```

The script then transmits the password "cs4552" along with the necessary carriage return and paused for the proper router response, "Router2#." The #-symbol is Cisco's default indicator that the router has entered the privileged mode.

```
haTypeText ("cs4552"&cr);
```

```
haWaitForPrompt ("Router2#");
```

The command to enter the configuration mode using a terminal was then entered and the proper router response was anticipated.

```
haTypeText ("config t"&cr);
```

```
haWaitForPrompt ("config)#");
```

The "show queuing" command is then sent, followed, as is the case with each command, by a carriage return.

```
haTypeText("sh queueing"+cr);
```

The function, `haWaitForString`, checks characters read from the connected device to verify they matched a specified character string. In this case, the string "configuration" is required because it indicates that the router has responded with the requested configuration status information.

```
haWaitForString ("configuration:");
```

The function `haGetInput` takes data received from the remote system after the command `haWaitForString`. The received character string is then loaded in the variable `vtr`.

```
vtr = haGetInput (1, 290, 100, 100);
```

The received information is then displayed to the user using a dialog box.

```
haMessageBox ("Queue info: ", vtr, 1);
```

The command "exit" is sent to leave the configuration mode and resent to close the Console port.

```
haTypeText ("exit"&cr);
```

The script exits with a call to the function,

```
haTerminate();
```

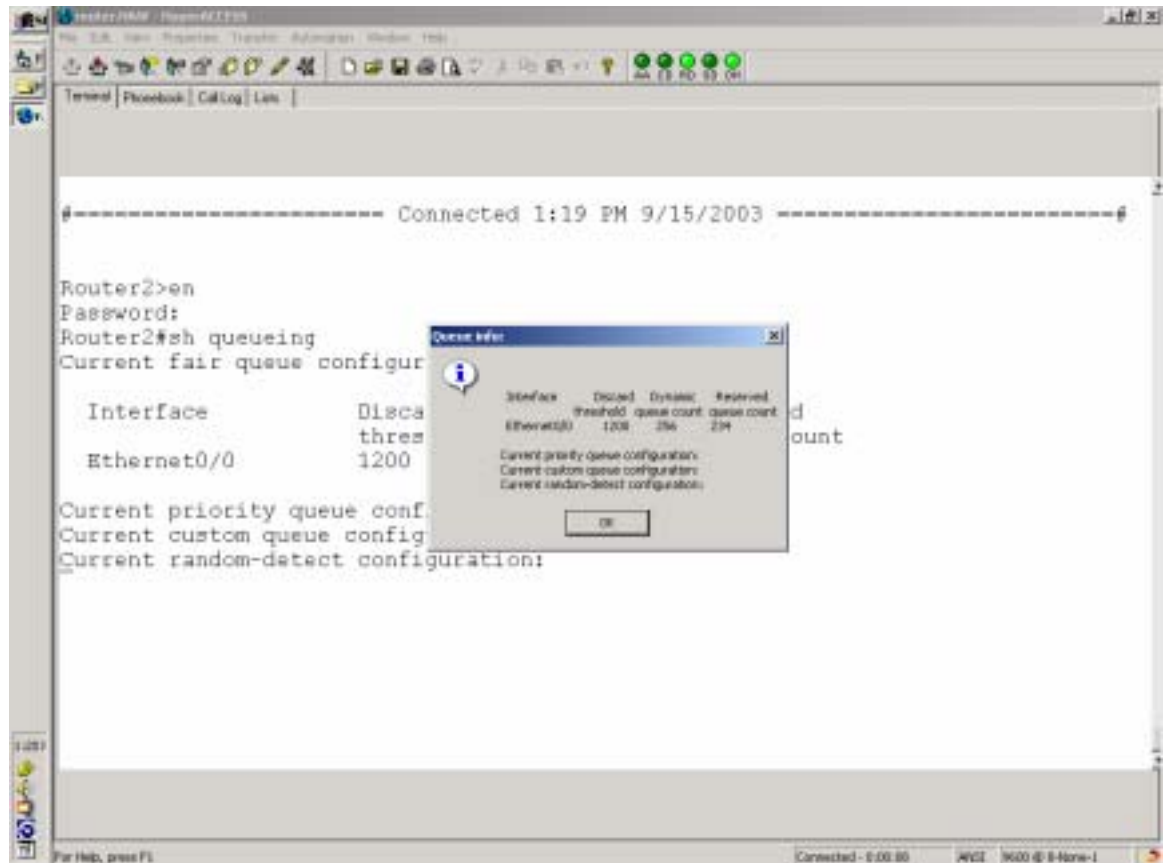


Figure 9. HyperACCESS Linked with the Router Displaying the Results of the Queue Status Query

The information that appears on the screen of the console terminal when the JavaScript is executed is shown in Figure 8. It supports the hypothesis that using dynamic scripts objects with HyperACCESS can extract all the information from the router required to verify the SAAM-directed configuration.

C. CUSTOM-WRITTEN PROGRAMS

Special security considerations or corporate implementations may require a customized user interface. Custom applications may also require a specific communication interface, or a more capable programming language to be developed.

As previously recognized, HyperACCESS is a software utility that provides access to any OLE automation, enabling external programming languages to use the available methods in the HyperACCESS API, and HAPI. The OLE automation allows developers, familiar with another language or compiler, such as Visual Basic and C++, to integrate and develop their own applications programs and to utilize the HyperACCESS communications utility, through the HAPI without learning a new programming tool.

Many development environments include instructions on how to call Windows API functions. Since HAPI is similar to these functions calls, everything that is applicable to the Windows API is applicable to the HAPI.

Every external program accessing HyperACCESS must create a script object. In order for the script object to be used, the OLE Automation is essential in the external program. The external program accesses an automation server, such as HyperACCESS, through a standard set of functions, called a dispatch interface. An automation client can call any function that is exposed through the dispatch interface. HyperACCESS implements two dispatch interfaces, the HyperACCESS interface and the HAScript

interface. Currently, the only purpose of the HyperACCESS interface is to return HAscript interfaces to the automation client.

Some steps need to be completed to automate the functions in HyperACCESS. First, the program has to obtain the active instance of HyperACCESS or to create a new instance of HyperACCESS. Second, it must query the active HyperACCESS instance for the HyperACCESS interface. Third, the function, `haInitialize`, which returns a HAscript interface, must be called. Then the program can call HAPI functions through this interface. When the program is finished with its tasks, it must call the function, `haTerminate`, to end its session, and then must release the interfaces, HAscript and HyperACCESS.

1. Microsoft Visual Basic

The programming language, Visual Basic, offers excellent support for OLE automation, principally because Microsoft based on the functionality of Visual Basic created the OLE. In order for a Visual Basic program to use the functions of HyperACCESS, it needs to declare the objects that will be used, obtained or created the active instance of HyperACCESS, obtain the script object, and initialize the object. When finished with the calls to the HAPI methods, the program must call the function, `haTerminate`, to close properly.

The Visual Basic for Applications environment provides utilities to create programs associated with other Microsoft application software, such as Access and Excel, which can use HyperACCESS to download information.

The example software, in Figure 9, shows Visual Basic code using HAPI functions to connect with the router and receive information about the state of the router's queues. The library, hawin32, shown in the object browser of the Visual Basic IDE demonstrates that the OLE automation was activated to provide access to the functions of HyperACCESS.

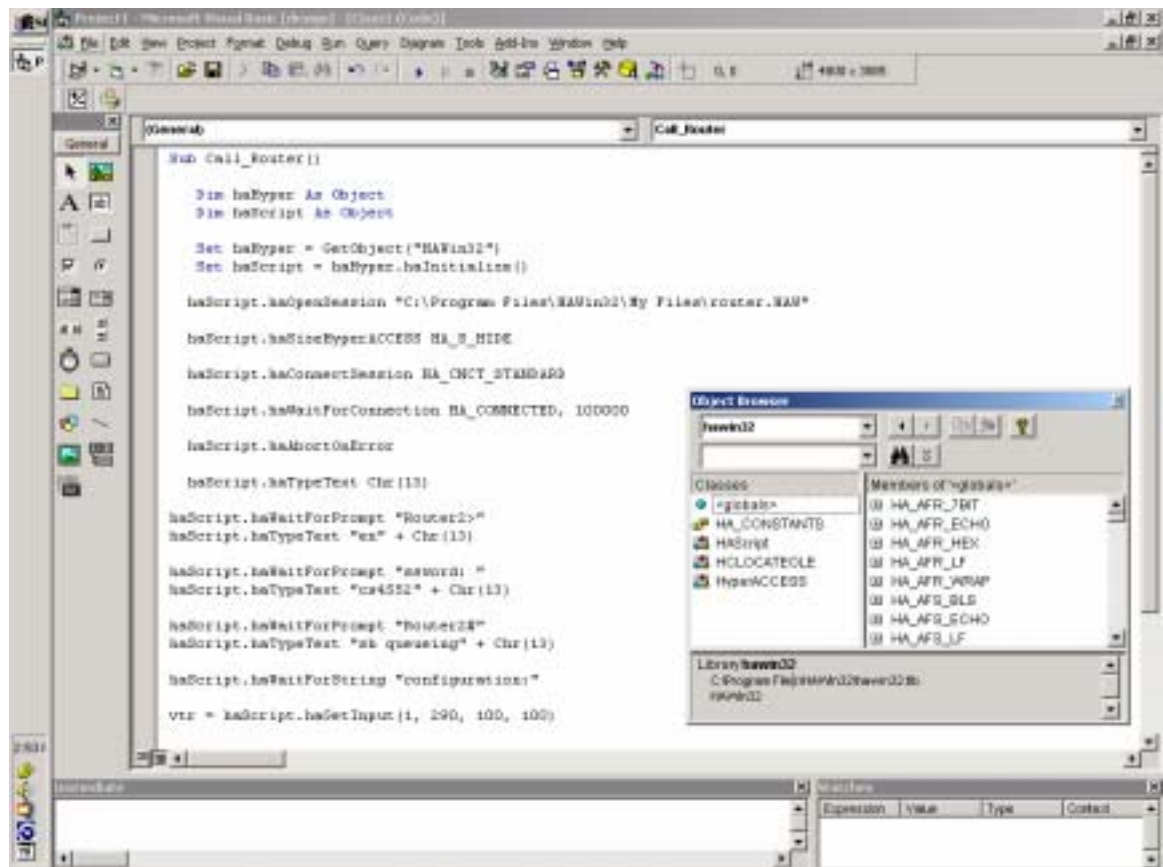


Figure 10. Visual Basic Accessing the HyperACCESS Library

a. The Code in Visual Basic

Following is an explanation of the key components of the Visual Basic code. As with the other code examples, the full code can be found in the appendices.

The HyperACCESS interface objects must be declared and initialized. Here it is assumed that a HyperACCESS instance is not already running; otherwise, the function, getObject() would be called instead of CreateObject ().

```
Dim haHyper As Object  
Dim haScript As Object  
Set haHyper = CreateObject ("HAWIN32")  
Set haScript = haAuto.Initialize ()
```

The program then opens the file router.HAW that is a script associate for use by HyperACCESS.

```
haScript.haOpenSession "C:\Program  
Files\HAWin32\MyFiles\routerv.HAW"
```

The HyperACCESS display to the screen is then hidden so that monitor updates do not delay the program execution.

```
haScript.haSizeHyperACCESS HA_S_HIDE
```

The HyperACCESS interface then connects to the open session and waits for the connection to be established. A time-out is specified to prevent a program delay in the event the connection cannot be established.

```
haScript.haConnectSession HA_CNCT_STANDARD  
haScript.haWaitForConnection HA_CONNECTED,  
100000
```

As with the scripting examples, any error automatically aborts the application.

```
haScript.haAbortOnError
```


Once the connection is established with the router through HyperACCESS, a carriage return must be transmitted to open the console-port session and the application must be paused pending receipt of the proper router response. Again, a time-out is established by the pause command to prevent the application from pausing indefinitely.

```
haScript.haTypeText Chr(13)
```

```
haScript.haWaitForPrompt "Router2>"
```

The command "en" and the carriage return are sent as a string of characters directing the router to enter the privileged mode.

```
haScript.haTypeText "en"& Chr(13)
```

```
haScript.haWaitForPrompt "ssword: "
```

Upon receipt from the router of the prompt for a password, the appropriate password string is sent and the application is paused for verification that the router indeed entered the privileged mode, as indicated by receipt of the proper prompt string.

```
haScript.haTypeText "cs4552"& Chr(13)
```

```
haScript.haWaitForPrompt "Router2#"
```

The character string, "config t" and the carriage return, are transmitted to place the router in the "configure router via terminal" mode. The receipt of the proper prompt verifies the mode.

```
haScript.haTypeText "config t"& Chr(13)
```

```
haScript.haWaitForPrompt "config)#"
```

The function, `haTypeText`, then forwards the command to the router to show the queuing status.

```
haScript.haTypeText "sh queueing"+ Chr(13)
```

The function, `haWaitForString`, checks characters read from the connected device to see if they match a specified character string. In this case, the string "configuration" is expected from the router.

```
haScript.haWaitForString "configuration:"
```

The function, `haGetInput`, extracts character data received from the remote system. The data is then loaded in the variable, `vtr`. The parameters 1, saying for processing backspaces, 290 is the number of characters to wait for, 100 is the timeout value, measure in milliseconds, the last 100 is the timeout, measure in milliseconds, which determines how long to wait between characters.

```
vtr = haGetInput (1, 290, 100, 100)
```

The results are then displayed in a dialog box to the user on the screen.

```
haScript.haMessageBox "Queue info: ", vtr, 1
```

The command "exit" is entered to return to the privilege mode and the application pauses pending receipt of the appropriate router response.

```
haScript.haTypeText "exit"& Chr(13)
```

```
haScript.haWaitForPrompt "Router2#"
```

The command "exit" is entered again to close the console port session.

```
haScript.haTypeText "exit"& Chr(13)
```

The script concludes the link with a call to the terminate function.

```
haScript.haTerminate
```

2. Microsoft Visual C++

To work with OLE Automation, the C++ software must to have a compiler that supports the Microsoft Foundation Classes. Therefore, the Visual C++ IDE was selected, since it supports OLE Automation at its most primitive level. A software application developed in Visual C++ will represent the SAAM Proxy Agent, using the console port.

The Visual C++ IDE offers limited support for using the OLE Automation when compared to Visual Basic. Therefore, developing some procedures is necessary to allow the use of the HyperACCESS haScript interface as used by the preceding Visual Basic example. The generation of wrapper classes for the haScript interface is essential. These classes have member functions that perform the actual function and call to the dispatch interface. When a member function is called, the called function looks like an ordinary C++ class member function.

A globally unique identifier (GUID) must also be created because the OLE Automation needs to specify each automation server. The programmer using the program GUIDGEN.EXE creates the GUID. The returned identification number is guaranteed to be unique for that moment in time. The GUIDs for the HyperACCESS Class ID is 5178CCE0-AAEF-11CE-AE75-00AA0030EBC8 and the HyperACCESS Interface ID is 5178CCE2-AAEF-11CE-AE75-00AA0030EBC8. These numbers are

unique, and they are not generated each time the program executes.

To complete the tasks necessary to permit the Visual C++ IDE to work with OLE Automation, should be implemented the Unknown Interface. This interface lets external programs query an object from an interface the Unknown Interface supports, such as IHyperACCESS or IHAScript.

The previous actions enable the Visual C++ IDE to use the functions of HyperACCESS. Any developed application code must point to the created script object and include the file, ha_auto.h, which is in the HyperACCESS directory.

V. CONCLUSION

A. THE PROBLEM

Day by day, individuals increasingly rely on their networks to run critical applications, such as video and telephony. This fact highlights the need for the network to function efficiently. To address this need, Quality of Service requirements for network service levels were defined. These requirements underscore the idea that a service has the goal of enabling the reliability, availability, and performance of the network.

The characteristics of network services vary by different application needs. Therefore, many kinds of services need to be described to adequately support the user requirements.

The most important device that interconnects networks is the router, which may monitor all the aspects that are extremely important in providing an adequate end-to-end communication system. These aspects include bandwidth availability, packet loss rates, and delivery delays through the network. It is perfectly viable to draw a parallel between a router's functionality and the Quality of Service provided.

The problem addressed by this research begins here since current network conditions and user requirements determine the values of Quality of Service parameter thresholds whether and those thresholds being exceeded. These parameters are dynamic, and they must be responsively uploaded to the Cisco routers so that the new values affect a change in the network performance. Not requiring any

modification to the router IOS for such uploads is preferable because the IOS is proprietary software. While such changes to the IOS are necessary to allow autonomous, dynamic configuration changes by the router, it is time-consuming and potentially costly to obtain the IOS source code from Cisco or arrange for a special IOS update release. For this reason, it was the tendency is to develop a means of using the Server and Agent-Based Active Network Management system (SAAM) to control the runtime configuration of Cisco routers functioning within the SAAM-enabled network. Dynamic, responsive management of network resources, based on communication between the SAAM Server, the SAAM capable routers, and SAAM proxies controlling non-SAAM-enabled routers is essential. This insures necessary dynamic resource management without proprietary software changes.

A main goal of the SAAM system is to optimize allocation of network resources to support the users' Quality of Service requirements. This thesis demonstrated several key components of a solution to integrate the SAAM system with a legacy network.

B. SOLVING

A SAAM proxy must communicate with its associated router and other proxies in a timely and secure manner. This thesis focused on the means to achieve that goal. The first step was to understanding those aspects of the Cisco IOS that were configured and/or controlled externally by a SAAM proxy. The next step was to develop automated methods of uploading the SAAM specified QoS parameter values to the

controlled Cisco router, such as flow definitions implemented by router interface configurations and reservations as well as routing table entries. Additionally critical was a means of extracting the current state information from a Cisco router. The final step was to develop a reliable means of communicating between the SAAM proxies.

A Cisco router typically has two ports for serial connections with an external control terminal, and the Console and Auxiliary (AUX) Ports. These ports provide the only means of direct access to the Command Line Interface (CLI) of the router's IOS. In addition, the AUX port can be accessed via a modem to permit remote configuration by way of dial-up communication. Securing a console connection is much easier than that of a dial-up connection. However, the console port is designed for manual, interactive communication. Therefore, a software utility must be developed to automate the console port access by a SAAM proxy, thereby eliminating the need for any system administrator intervention.

After the initial research, I concluded that HyperACCESS is a good software platform for automating the communication between a SAAM proxy and its router. HyperACCESS provides an Application Programming Interface (API) that can be called from JavaScript or VBScript programs. The API may also be integrated with code written in other programming languages, such as C++ and Visual Basic. The API provides access to library methods that allow the console port session to be manipulated by an application program rather than just by a person.

Since the SAAM proxies and their associated routers are in the same TCP/IP network, the SAAM proxies may also remotely communicate with the routers via the TELNET protocol. A proxy can use a script, written in a language like Perl, to open and maintain a TELNET connection with a Cisco router. The TELNET access should be secured with the Secure Shell (SSH) protocol.

The Simple Network Management Protocol (SNMP) provides another alternative for extracting state information from a Cisco router. However, previous studies concluded that it is impossible to use that protocol to upload some of the SAAM specific configurations into a router. The SNMP may be a better choice for extracting router state information than the Cisco IOS Command Line Interface because the latter option requires parsing of complex command output text strings.

C. FUTURE WORK

The demonstration of the sample scripts for uploading configuration settings, collecting interface statistics, and establishing communications between proxy agents easily indicates the potential for using script languages, such as Perl, JavaScript, and VBScript, to monitor, manage, and control the configuration of Cisco routers. These demonstrations should be expanded to validate the concept over a more demanding network, using traffic generators to stress the router interfaces.

In order to verify the assertion that custom applications can be developed using high order languages to satisfy security or organizational requirements, a C++,

Visual Basic, or Java-based application should be developed to stress the communication used in the SAAM system between the Server, the enabled routers, and the proxy agents. This application should generate Cisco configuration changes by the proxies based on messages received from the SAAM Server regarding flow allocations.

This kind of communication technique could support not only Quality of Service implementations but also many solutions that could be used to economize the usage of resources. This technique will also improve the use of existing information system resources. The Cisco router was chosen to demonstrate communications between a SAAM proxy and a proprietary router, but other devices or brands of routers could be used. The only requirement would be to understand the operating system software functionality of the targeted device.

Worth mentioning also is that products like Cisco routers are becoming very complex, as they evolve to meet the needs of the individuals and corporations utilizing the networks. The prices are decreasing and becoming very attractive. This price drop could be an obstacle for implementing solutions that use only SAAM-enabled devices. Therefore, it is prudent to find efficient solutions for dynamic communication between networking devices, since these solutions may find usefulness beyond just SAAM-based implementations. Doing so ensures they can be separated from the SAAM system and be utilized for many proposes other than just for QoS enforcement. Such solutions insure a timely, automatic and secure way of communicating between network devices.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

1. Network Programming with Perl- Lincoln d. Stein.
2. NET::TELNET::CISCO PERL module - Appendix B.
3. CCNA - Cisco Press; Wendell Odom.
4. Configuring CISCO Routers using Telnet, Key Ehresman.
5. Experiment with CISCO MIBs, QoS, and SNMP Features for Adapting SAAM Flow/Path Based Routing and Control, Cary Colwell.
6. Programming CISCO Router using SNMP, Spring/2002 CS4552 Project Report -Valter Monteiro Jr.
7. James M. Lacey, Visual C++ 6 Distributed, Exam Cram.
8. HyperACCESS for Windows Version 8.3, Application Programming Interface Manual.
9. Fatih Turksoyu, "Realistic Traffic Generation Capability for SAAM Testbed", Thesis, March 2001, Computer Science Department Naval Postgraduate School.
- 10.<http://www.cisco.com>. Jul/2003
- 11.<http://www.cisco.com/univercd/cc/doc/products/software/ios121/121cgcr/funr/frprt3/frd3003.html#xtocid2298129>
Jul/2003
- 12.<http://www.cisco.com/warp/public/471/mod-aux-exec>
Jul/2003
- 13.http://www.cisco.com/en/US/products/sw/iosswrel/ps1834/products_feature_guide09186a Jul/2003

- 14.<http://www.perldoc.com> Jul/2003
- 15.<http://perl.org> Aug/2003
- 16.<http://www.1024kb.net/texts/perlner.html> Aug/2003
- 17.<http://nettelnetcisco.sourceforge.net> Aug/2003
- 18.<http://cpan.org> Aug/2003
- 19.<http://open-Perl-ide.sourceforge.net> Aug/2003
- 20.<http://perlflect.com/articles/sockets.shtml> Aug/2003
- 21.<http://computer.howstuffworks.com/perl.htm> Aug/2003
- 22.<http://theoryx5.uwinnipeg.ca/CPAN/data/Net-Telnet/Net/Telnet.html> Aug/2003
- 23.<http://www.strom.com/pubwork/hyperaccess.html>
Sept/2003

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
Ft. Belvoir, VA
2. Dudley Knox Library
Naval Postgraduate School
Monterey, CA
3. Oswaldo Zaneli
DCEA
Rio de Janeiro, Brazil
4. Prof. Peter J. Denning, Chairman, Code CS
Naval Postgraduate School
Monterey, CA
5. Prof. Geoffrey Xie, Code CS/Xg
Naval Postgraduate School
Monterey, CA
6. Prof. John Gibson, Cod CS/LN
Naval Postgraduate School
Monterey, CA